IRIX® Interactive Desktop Integration Guide

CONTRIBUTORS

Written by Beth Fryer, Jed Hartman, Ken Jones, and Pete Sullivan

Updated by Max Anderson

Updated by Julie Boney

Illustrated by Beth Fryer and Seth Katz

Production by Karen Jacobson

Engineering contributions by Bob Blean, Susan Dahlberg, Susan Ellis, John Krystynak, Chandra Pisupati, Jack Repenning, CJ Smith, Dave Story, Steve Strasnick, Rebecca Underwood, Steve Yohanan, and Betsy Zeller.

COPYRIGHT

© 1999, 2001 Silicon Graphics, Inc. All rights reserved; provided portions may be copyright in third parties, as indicated elsewhere herein. No permission is granted to copy, distribute, or create derivative works from the contents of this electronic documentation in any manner, in whole or in part, without the prior written permission of Sili

LIMITED RIGHTS LEGEND

The electronic (software) version of this document was developed at private expense; if acquired under an agreement with the USA government or any contractor thereto, it is acquired as "commercial computer software" subject to the provisions of its applicable license agreement, as specified in (a) 48 CFR 12.212 of the FAR; or, if acquired for Department of Defense units, (b) 48 CFR 227-7202 of the DoD FAR Supplement; or sections succeeding thereto. Contractor/manufacturer is Silicon Graphics, Inc., 1600 Amphitheatre Pkwy 2E, Mountain View, CA 94043-1351

TRADEMARKS AND ATTRIBUTIONS

Silicon Graphics, IRIS, IRIX, and OpenGL are registered trademarks and SGI, IRIS GL, the SGI logo, IRIS InSight, IRIS ViewKit, the Graphics Library, Open Inventor, and RapidApp are trademarks of Silicon Graphics, Inc. Apple is a registered trademark and Apple Quicktime is a trademark of Apple Computer, Inc. Kodak is a trademark of Eastman Kodak Company. Microsoft is a registered trademark of Microsoft Corporation. Motif and OSF/Motif are trademarks of the Open Software Foundation. PostScript is a registered trademark of Adobe Systems, Inc. Solaris is a registered trademark and Sun and SunOS are trademarks of Sun Microsystems, Inc. UNIX is a registered trademark of the Open Group in the United States and other countries. Wavefront is a trademark of Alias/Wavefront, a division of Silicon Graphics Limited. X Window System is a trademark of the Open Group.

Cover Design By Sarah Bolles, Sarah Bolles Design, and Dany Galgani, SGI Technical Publications.

Contents

	Examples
	Figures
	Tables
	New Features in This Guide
	Record of Revision
	About This Guide
	What This Guide Contains
	How to Use This Guide
	What You Should Know Before Reading This Guide
	Suggested Reading
	Font Conventions in This Guide
	Integrating an Application Into the IRIX Interactive Desktop Environment:
	integrating an Application into the IRIX interactive Desktop Environment.
	An Introduction
	An Introduction
PART I	An Introduction
PART I	An Introduction
	An Introduction

007-2006-130 iii

2.	Getting the IRIX Interactive Desktop Look	5
3.	Using Schemes	7
	Schemes Overview	7
	Why You Should Use Schemes	8
	Basic Scheme Concepts	8
	Using Schemes in Your Application	9
	Turning on Schemes for Your Application	9
	Special Considerations for Programming with Schemes	10
	Assigning Non-Default Colors and Fonts to Widgets	11
	Directly Accessing Colors and Fonts	13
	Pre-Defined Scheme Resources and Symbolic Values	15
	Extending a Scheme to Support New Colors	18
	Scheme File Organization.	19
	How to Extend a Scheme	19
	Testing Your Application with Schemes	21
	Creating New Schemes	21
	Hard-Coding a Scheme for an Application	22
4.	Using the SGI Enhanced Widgets	23
	Using the SGI Enhanced Widgets	23
	Using the Widget Demos	24
	Location of Widget Demos	24
	Instructions for Building the Widget Demos	24
	The Enhanced Widgets	
	The File Selection Box Widget	25
	The Scale (Percent Done Indicator) Widget	
	The Text and TextField Widgets	
	The Mixed-Model Programming Widgets	

iv 007-2006-130

	The SGI Enhanced Widgets)
	The Color Chooser Widget	J
	The Dial Widget	5
	The Thumbwheel Widget	3
	The Drop Pocket Widget	1
	The Finder Widget	2
	The Graph Widget	5
	The Springbox Widget	7
	The Grid Widget	3
	HTML Viewer Component	2
	Overview of the HTML Viewer Components	4
	Viewer Components	4
	Supported Tags and Attributes	5
5.	Window, Session, and Desk Management	9
	Window, Session, and Desk Management Overview	
	Window Management	
	Session Management	J
	Desk Management	1
	Further Reading on Window and Session Management 69	5
	Implementing an Application Model	
	Implementing the "Single Document, One Primary" Model 66	5
	Implementing the "Single Document, Multiple Primaries" Model 66	5
	Implementing the "Multiple Document, Visible Main" Model 66	5
	Implementing the "Multiple Document, No Visible Main" Model 67	7
	Interacting With the Window and Session Manager	3
	Creating Windows and Setting Decorations	3
	Handling Window Manager Protocols	5
	Setting the Window Title	9
	Controlling Window Placement and Size	9
6.	Customizing Your Application's Minimized Windows.	3
	Some Different Sources for Minimized Window Images	

007-2006-130 v

	Creating a Minimized Window Image: The Basic Steps									. 85
	Using mediarecorder to Get an RGB Format Image									. 86
	Resizing the RGB Image Using imgworks									. 88
	Setting the Minimized Window Label									. 89
	Changing the Minimized Window Image									. 89
7.	Interapplication Data Exchange									. 91
	Data Exchange Overview									. 91
	Primary Transfer Model Overview									. 92
	Clipboard Transfer Model Overview									. 93
	Interaction Between the Primary and Clipboard Tran	nsfe	er N	Iod	els					. 94
	Implementing the Primary Transfer Model									. 95
	Data Selection									. 95
	Requests for the Primary Selection									. 96
	Loss of the Primary Selection									. 97
	Inserting the Primary Selection									. 97
	Implementing the Clipboard Transfer Model									. 98
	Cut Actions									. 98
	Copy Actions									100
	Requests for the Clipboard Selection									100
	Paste Actions									100
	Loss of the Clipboard Selection									101
	Supported Target Formats									101
	Data Conversion Service									105
	The Converter Registry									106
	The GoldenGate API									106
8.	Monitoring Changes to Files and Directories									107
	FAM Overview									107
	Theory of Operation									108
	FAM Libraries and Include Files.								_	108

vi 007-2006-130

	The FAM Interface)9
	Opening and Closing a FAM Connection)9
	Monitoring a File or Directory	10
	Suspending, Resuming, and Canceling Monitoring	11
	Detecting Changes to Files and Directories	12
	FAM Examples	16
	Using FAM	18
	Waiting for File Changes	18
	Polling for File Changes	22
9.	Providing Online Help With SGIHelp	25
	Overview of SGIHelp	26
	The Help Viewer	26
	The SGIHelp Library and Include File	28
	Help Document Files	<u> 2</u> 9
	Application Helpmap Files	29
	The SGIHelp Interface	29
	Initializing the Help Session	30
	Displaying a Help Topic	31
	Displaying the Help Index	32
	Implementing Help in an Application	33
	Constructing a Help Menu	34
	Implementing a Help Button	35
	Providing Context-Sensitive Help	36
	Application Helpmap Files	38
	Helpmap File Conventions	39
	Helpmap File Format	39
	Widget Hierarchies in the Helpmap File	12

007-2006-130 vii

	Writing the Online Help
	Overview of Help Document Files
	Viewing the Sample Help Document Files
	Creating a Help Document File
	Preparing to Build the Online Help
	Building the Online Help
	Finding and Correcting Build Errors
	Producing the Final Product
	Creating the Installable Subsystem
	Incorporating the Help Subsystem into an Installable Product
	Incorporating the Help Subsystem into a Product With a Custom Installation Script 15.
	Bibliography of SGML References
10.	Handling Users' System Preferences
	Handling the Mouse Double-Click Speed Setting
	Using the Default Viewer and Editor Utilities Panel
	Selecting Utilities and Their Values
	Setting the Preferred Text Editor
PART II	Creating Desktop Icons
11.	Creating Desktop Icons: An Overview
	About IRIX Interactive Desktop Icons
	Checklist for Creating an Icon
	Creating an Icon: The Basic Steps Explained in Detail
	Step One: Tagging Your Application
	Step Two: Drawing a Picture of Your Icon
	Step Three: Programming Your Icon
	Step Four: Compiling the Source Files
	Step Five: Installing Your Application in the Icon Catalog
	Step Six: Restarting the Desktop
	Step Seven: Updating Your Installation Process
12.	Using IconSmith
	About IconSmith

viii 007-2006-130

Where to Install Your Completed Ic	on											.178
Some Definitions												.179
Caret												.179
Transformation Pin												.179
Vertex												.180
Path												.180
Starting IconSmith												.180
IconSmith Menus												.181
IconSmith Windows												.182
Drawing With IconSmith												.183
Drawing Paths												.185
Drawing Filled Shapes												.185
Deleting												.186
Keeping the 3-D Look												.187
Drawing for All Scales												.187
Sharing Design Elements .												.188
Templates												.189
Selecting												.189
Partial												.190
Deselect Fragments												.190
Select Next												.191
Select All												.191
Transformations												.191
Scale												.192
Scale X and Y \dots												.192
Rotate												.193
Shear Y												.193
Concave Polygons												.193
Constraints: Gravity (Object) Snap a	ınd	Gr	id S	Sna	p							.194
Controlling the Grid												.195
Controlling Gravity												.195

007-2006-130 ix

	Icon Design and Composition Conventions
	Importing Generic Icon Components (Magic Carpet)
	Icon Size
	Selecting Colors
	Advanced IconSmith Techniques
	Drawing a Circle
	Drawing an Oval
	Isometric Circles
13.	File Typing Rules
	A Table of the FTRs With Descriptions
	Naming File Types: The TYPE Rule
	Categorizing File Types: The SUPERTYPE Rule
	Matching File Types With Applications: The MATCH Rule
	Matching Tagged Files
	Matching Files Without the tag Command
	Matching Non-Plain Files: The SPECIALFILE Rule
	Adding a Descriptive Phrase: The LEGEND Rule
	Setting FTR Variables: The SETVAR Rule
	Programming Open Behavior: The CMD OPEN Rule
	Programming Alt-Open Behavior: The CMD ALTOPEN Rule
	Programming Drag and Drop Behavior: The CMD DROP and DROPIF Rules
	Mapping Names: The MAP Rule
	Programming Print Behavior: The CMD PRINT Rule
	Adding Menu Items: The MENUCMD Rule
	Getting the Icon Picture: The ICON Rule
	Creating a File Type: An Example
	Open an FTR File for scribble
	Add the Rules to the scribble FTR File
	Name the scribble FTR File and Put It in the Appropriate Directory
	The scribble FTRs
14.	Printing From the Desktop
	About routeprint

x 007-2006-130

	Converting a File for Printing
	Print Costs
	The Print Conversion Rules
	The CONVERT Rule
	The COST Rule
	The FILTER Rule
	The Current Printer
A.	Example Programs for SGI Enhanced Widgets
	Example Program for Color Chooser
	Makefile for colortest.c
	Example Program for Dial
	Example Program for Drop Pocket
	Makefile for Drop Pocket Example
	Example Program for Finder
	Example Program for History Button (Dynamenu)
	Example Program for ThumbWheel
	Example Program for File Selection Box
	Makefile for File Selection Box Example Program
	Example Programs for Scale (Percent Done Indicator) Widget
	Example Program for LED Widget
В.	Desktop Variables
	Variables Set By the Desktop
	Variables Set By the User
C	
C.	Online Help Examples
	1 1
	Allowable Elements in a Help Document
D.	The Icon Description Language
	Operators
	Constants
	Variables
	Functions 30°

007-2006-130 xi

E.	Predefined File Types	309
	Naming Conventions for Predefined File Types	309
	The Predefined File Types and What They Do	310
	SpecialFile	310
	Directory	310
	Ascii	310
	Source Files	311
	Binary	311
	ImageFile	312
	Executable	312
	Scripts	313
	GenericWindowedExecutable	313
	LaunchExecutable	313
	ttyExecutable	314
	ttyLaunchExecutable	314
	ttyOutExecutable	314
	ttyLaunchOutExecutable	315
F.	FTR File Directories	317
G.	Using GoldenGate Data Conversion Services	319
	-	319
		320
	Selecting a Converter	322
		328
		338
		339
	<u>e</u>	339
	· · · · · · · · · · · · · · · · · · ·	340
	· · · · · · · · · · · · · · · · · · ·	347
	Testing Your Converter	351
		352
		353
	· · · · · · · · · · · · · · · · · · ·	353

xii 007-2006-130

H.	Standard Menu Resources .										.359
	Common Menu Bar Resources										.359
	Standard File Menu Resources										.360
	Standard Edit Menu Resources										.361
	Index										.363

007-2006-130 xiii

Examples

Example 3-1	Retrieving a Scheme Color Value	. 13
Example 4-1	An Example of Using the Grid Widget	
Example 4-2	Another Example of Using the Grid Widget	. 51
Example 5-1	Session Management Example Code: saveyourself.c	. 63
Example 5-2	Creating a Main Primary Window	. 69
Example 5-3	Creating a Co-Primary Window	. 71
Example 5-4	Creating a Support Window	. 73
Example 5-5	Creating a Dialog	. 74
Example 5-6	Handling the Window Manager Quit Protocol	. 75
Example 5-7	Handling the Window Manager Delete Window Protocol in Co-Primary Windows	. 76
Example 5-8	Handling the Window Manager Delete Window Protocol in Support Windows and Dialogs	. 77
Example 5-9	Handling the Window Manager "Save Yourself" Protocol	. 78
Example 7-1	Asserting Ownership of PRIMARY Selection	. 95
Example 7-2	Handling Cut Actions in the Clipboard Transfer Model	. 98
Example 8-1	Using Select With FAM	.119
Example 8-2	Polling With FAM	.122
Example 8-3	Polling FAM Within an Xt Work Procedure	.123
Example 9-1	Initializing a Help Session Using SGIHelpInit()	.130
Example 9-2	Requesting a Specific Help Topic Using SGIHelpMsg()	.132
Example 9-3	Requesting a Help Topic for a Widget Using SGIHelpMsg() .	.132
Example 9-4	Displaying a Help Index Using SGIHelpIndexMsg()	.133
Example 9-5	Providing a Help Button	.135
Example 9-6	Implementing Context-Sensitive Help	.136
Example C-1	An Example of a Help Source File	.266
Example C-2	A Description of the Elements Defined by the Help DTD	.274

007-2006-130 xv

Example C-3	An Example of Integrating SGIHelp With an Application .		289
Example C-4	Help Source File for Example Program		300
Example C-5	Helpmap for Example Program		304

xvi 007-2006-130

Figures

Figure i	The IRIX Interactive Desktop x	xxi
Figure 4-1	The File Selection Box Widget	25
Figure 4-2	The Color Chooser Widget	31
Figure 4-3	The Color Chooser Widget with HSV and RGB Sliders	32
Figure 4-4	The Dial Widget in Knob and Pointer Form	36
Figure 4-5	The Thumbwheel Widget	38
Figure 4-6	The Drop Pocket Widget (on left) As Part of the Finder Widget	41
Figure 4-7	The Finder Widget	43
Figure 4-8	The Graph Widget	46
Figure 4-9	The HTML Widget	53
Figure 5-1	Window Settings Control Panel	61
Figure 6-1	Minimized Window Image Examples	84
Figure 6-2	The mediarecorder Tool	86
Figure 9-1	The Help Viewer	127
Figure 9-2	The Help Index Window	128
Figure 10-1	The dtUtilities Panel	156
Figure 12-1	Caret	179
Figure 12-2	Transformation Pin	180
Figure 12-3	Vertex	180
Figure 12-4	Path	180
Figure 12-5	The IconSmith Icon	180
Figure 12-6	The Main IconSmith Window With Popup Menus	182
Figure 12-7	The Import Icon or Set Template Window	184
Figure 12-8	The Palette (Selection Properties) Window	186
Figure 12-9	3-D Icon Axes	187
Figure 12-10	Concave Polygon	193
Figure 12-11	The Constraints Window	194

007-2006-130 xvii

Figure 12-12	A	Path																198
Figure 12-13	W	heel S	pok	es														199
Figure 12-14	Co	nnect	ted S	Spok	es													199
Figure 12-15	Fir	nished	d 2-E	O Cir	cle													199
Figure 12-16	Ar	n Ova	1.															200
Figure 12-17	A	Simpl	le, C	ircul	ar 2	2-D	Ico	n										201
Figure 12-18	Im	porte	d Ci	rcles	3.													202
Figure 12-19	Fir	nished	d Iso	metı	ic (Circ	le											203
Figure 12-20	Sir	nple,	Ison	netri	c 2-	DΙ	con											203
Figure 12-21	Ico	on Cei	ntere	ed or	ı Ge	ene	ric (Con	npo	ner	ıt							204
Figure 12-22	Op	en Ic	on															205
Figure 14-1	Fil	e Cor	vers	sions	foi	r Pr	inti	ng S	Star	ıdaı	rd I)esl	ctor	o Fi	les			236

xviii 007-2006-130

Tables

Table i	Checklist of Implementation Tasks and References xxxi
Table ii	Tasks Requiring Application Changes and/or Motif xxxii
Table iii	Integration and ViewKit xxxiv
Table iv	Integration and RapidApp xxxv
Table 3-1	Pre-Defined Scheme Resources and Symbolic Values
Table 4-1	HTML Viewer Tags and Attributes
Table 4-2	HTML Viewer Character Tags
Table 7-1	Audio Formats
Table 7-2	Image Formats
Table 7-3	Movie Formats
Table 7-4	3D Graphics Formats
Table 7-5	Additional Data Types Supported by Silicon Graphics
Table 7-6	World Wide Web Targets
Table 10-1	dtUtilities and Their Values
Table 13-1	Rules That Appear in a Filetype Definition
Table 13-2	Numerical Representations in Match-Expressions
Table 13-3	Match-Expression Functions
Table 14-1	Conversion Costs for Print Conversion Rules
Table D-1	Icon Description Functions
Table E-1	Predefined File Type Naming Conventions
Table G-1	Converter Attributes
Table G-2	Query Operators
Table G-3	Converter Return Status Values
Table G-4	Converter Description File Statements

007-2006-130 xix

New Features in This Guide

This update of the IRIX Interactive Desktop Integration Guide supports the 6.5.14 release of IRIX.

Major Documentation Changes

A new example of the IRIX Interactive Desktop and descriptions of the windows appear in the Introduction.

007-2006-130 xxi

Record of Revision

Version	Description
130	November 2001
	Revision to support IRIX version 6.5.14

007-2006-130 xxiii

About This Guide

This book explains how to integrate applications into the IRIX Interactive Desktop environment. This book assumes that your applications run on Silicon Graphics workstations.

What This Guide Contains

This book is divided into two sections:

- Part One explains how to achieve the Silicon Graphics look and feel for your application. (Guidelines for look and feel are provided in the IRIX Interactive User Interface Guidelines.)
- Part Two explains how to create Desktop icons for your application and install them in the Icon Catalog.

How to Use This Guide

This book is a companion to the *IRIX Interactive User Interface Guidelines*. Silicon Graphics recommends that you read through the *IRIX Interactive User Interface Guidelines* first, then use the *IRIX Interactive Desktop Integration Guide* to help you implement the style guidelines.

What You Should Know Before Reading This Guide

This guide assumes that you are familiar with the material contained in the *OSF/Motif Style Guide* and the *IRIX Interactive User Interface Guidelines* manual. It assumes also that you have some knowledge of programming in IRIS IM and Xt (or Xlib).

007-2006-130 xxv

Silicon Graphics provides both these manuals online. You can view them from the IRIS InSight viewer. To use the IRIS InSight viewer, select "Online Books" from the Help toolchest.

Suggested Reading

Here are some books that provide information on some of the topics covered in this guide:

- *IRIS IM Programming Guide*. (This book is included online with the Silicon Graphics IRIS Development Option.)
- *IRIS ViewKit Programmer's Guide*. (This book is included online with the Silicon Graphics C++ option.)
- *OpenGL on Silicon Graphics Systems*. (This book is included online with the Silicon Graphics IRIS Development Option.)
- Open Software Foundation. *OSF/Motif Programmer's Guide, Revision 1.2*. Englewood Cliffs: Prentice-Hall, Inc., 1992. (This book is included online with the Silicon Graphics IRIS Development Option.)
- Open Software Foundation. OSF/Motif Style Guide, Revision 1.2. Englewood Cliffs: Prentice-Hall, Inc., 1992. (This book is included online with the Silicon Graphics IRIS Development Option.)
- Nye, Adrian and O'Reilly, Tim. The X Window System, Volume 4: X Toolkit Intrinsics Programming Manual, OSF/Motif 1.2 Edition for X11, Release 5. Sebastopol: O'Reilly & Associates, Inc., 1992. (This book is included online with the Silicon Graphics IRIS Development Option.)
- Nye, Adrian. *The X Window System, Volume 1: Xlib Programming Manual for Version 11 of the X Window System.* Sebastopol: O'Reilly & Associates, Inc., 1992. (This book is included online with the Silicon Graphics IRIS Development Option.)
- Young, Doug. *The X Window System, Programming and Applications with Xt, OSF/Motif Edition, Second Edition. Englewood Cliffs: Prentice Hall, Inc., 1994.*
- Assente & Swick. The X Toolkit.
- Scheifler, Robert and Gettys, Jim. *X Window System, Third Edition*. Digital Press, ISBN 1-55558-088-2.
- X/Open Company, Ltd. X/Open Portability Guide (set of 7 volumes). Englewood Cliffs: Prentice Hall Publishing Company, ISBN 0-13-685819-8.

xxvi 007-2006-130

To obtain SGI documentation, go to the SGI Technical Publications Library at:

http://techpubs.sgi.com

Font Conventions in This Guide

These style conventions are used in this guide:

- **Boldfaced text** indicates that a term is an option flag, a data type, a keyword, a function, a command-line option, or an X resource.
- "Quoted text" indicates menu items.
- Screen type shows code examples and screen displays.
- Bold screen type indicates user input and nonprinting keyboard keys.
- Regular text shows menu and window names, and X properties.

007-2006-130 xxvii

Integrating an Application Into the IRIX Interactive Desktop Environment: An Introduction

This book describes how to integrate your application into the IRIX Interactive Desktop environment. It assumes that your application already runs on Silicon Graphics workstations. This is strictly a how-to guide—refer to the IRIX Interactive Desktop User Interface Guidelines for style guidelines.

This introduction contains these sections:

- "About the IRIX Interactive Desktop Environment" presents a brief overview of the IRIX Interactive Desktop and explains why it's important to integrate your application into the Desktop environment.
- "Implementation Strategies and Toolkits" provides a checklist to help developers focus on the items that most benefit their users.
- "Integrating an Application" offers a brief, general list of the basic steps for integration.

About the IRIX Interactive Desktop Environment

The IRIX Interactive Desktop environment provides a graphical user interface (GUI) to the IRIX filesystem and operating system. This interface allows users to interact with the workstation using a point-and-click interface, based on icons and windows. The Desktop provides tools and services for the users' convenience, many of which are accessible directly from the Desktop's toolchests.

Integrating your application into the Desktop environment is an important step in creating your product. Since users are already familiar with the Desktop, they have certain expectations about how applications should look and behave in the Desktop environment. By integrating your application into the Desktop, you insure that these expectations are met—thus helping your users get the most out of your application.

007-2006-130 xxix

Figure i shows an example of the IRIX Interactive Desktop, introduced with IRIX version 6.5.14, 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

xxx 007-2006-130

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



Figure i The IRIX Interactive Desktop

007-2006-130 xxxi

Implementation Strategies and Toolkits

This section presents strategies for implementing your application and suggests some toolkits that will make the implementation easier. Topics include:

- "Implementation Checklist" on page xxxii
- "Using ViewKit and RapidApp" on page xxxiv

Implementation Checklist

Table i provides a checklist to help you focus your resources on the items that most benefit your users. The checklist lists tasks in order of importance. Try to adhere to the user interface guidelines in the order presented in the checklist.

For a summary of user interface guidelines that includes a complete checklist, see Appendix A, "Summary of Guidelines," in *IRIX Interactive Desktop User Interface Guidelines*.

Table i Checklist of Implementation Tasks and References

Task	IRIX Interactive Desktop User Interface Guidelines	Implementation Reference
Icons and File Typing Rules (FTRs)	Chapter 2	Desktop Integration Guide, Chapters 11-15
IRIX Interactive Desktop Look*	Chapter 3	Desktop Integration Guide, Chapters 2-3
Menus and Accelerators*	Chapter 3	OSF/Motif Programmer's Guide, Chapter 6
Copy and Paste*	Chapter 5	Desktop Integration Guide, Chapter 7
Window Management	Chapter 3	Desktop Integration Guide, Chapters 5-6
Software Installation	Chapter 4	S/W Packager User's Guide
Session Management	Chapter 3	Desktop Integration Guide, Chapter 5

xxxii 007-2006-130

Table i (continued)	Checklist of Implementation Tasks and References				
Task	IRIX Interactive Desktop User Interface Guidelines	Implementation Reference			
Online Help	Chapter 4	Desktop Integration Guide, Chapter 9			
Selection, Focus*	Chapter 7	OSF/Motif Programmer's Guide, Chapter 13			
Feedback	Chapter 11	OSF Motif Programmer's Guide			
Internationalization	Chapter 4	Topics in IRIX Programming, Chapter 4			

^{*} Items requiring use of the Motif toolkit.

It's also useful to know which changes you can make without modifying the application's source code and which items require the use of the IRIS IM (OSF/Motif) toolkit. The Silicon Graphics style is based on Motif, so using the Motif toolkit makes compliance much easier. Table ii lists which tasks require application code changes and which require Motif.

Table ii Tasks Requiring Application Changes and/or Motif

Task	Application Code Changes	Requires Motif
Icons and FTRs		
IRIX Interactive Desktop Look		yes
Menus/Accelerators		yes
Copy & Paste	yes	yes (Xt)
Window Management		
Software Installation		
Session Management	yes	
Online Help	yes Online help and context sensitive help require no code changes with ViewKit.	*

007-2006-130 xxxiii

Table ii (continued)	Tasks Requiring Application Changes and/or Motif			
Task	Application Code Changes	Requires Motif	Requires Motif	
Selection, Focus		yes		
Feedback	yes			
Internationalization	yes			

Using ViewKit and RapidApp

Besides using Motif, other toolkits and tools can make integrating your application easier. These include:

- "ViewKit"
- "RapidApp"

ViewKit

IRIS ViewKit is a C++ based, user-interface toolkit based on OSF/Motif. ViewKit also runs on Dec, HP, IBM, SCO, SunOS, and Sun Solaris.

Table iii lists integration tasks that you can achieve by using ViewKit.

Table iii Integration and ViewKit

Task	ViewKit
Icons and FTRs	
IRIX Interactive Desktop Look	Color schemes set by default
Menus and Accelerators	
Copy and Paste	VkCutPaste Class
Window Management	
Software Installation	
Session Management	Initial session management set
Online Help	Help menu entry and context sensitive help

xxxiv 007-2006-130

Table iii (continued)	Integration and ViewKit ViewKit	
Task		
Selection, Focus		
Feedback	Busy state and cursor are easily set	
Internationalization		

Your application can provide World Wide Web access by using VkWebViewerBase, which provides basic Web functionality. For more information, see the *IRIS ViewKit Programmer's Guide*.

007-2006-130 xxxv

RapidApp

Developer Magic RapidApp is an interactive tool for creating applications. It integrates with other Developer Magic tools, including cvd, cvstatic, cvbuild, and others, to provide an environment for developing object-oriented applications as quickly as possible. RapidApp generates C++ code, with interface classes based on the IRIS ViewKit toolkit. In addition to the conveniences provided by IRIS ViewKit, you can use RapidApp to help create your application (see Table iv).

Table iv Integration and RapidApp

Task	RapidApp	
Icons and FTRs	Generates an FTR rule and sample icon	
IRIX Interactive Desktop Look	sgiMode & schemes set by default	
Menus and Accelerators	Standard & Common menu entries	
Copy and Paste		
Window Management		
Software Installation	Automatically builds an inst image	
Session Management	Initial session management set	
Online Help		
Selection, Focus		
Feedback		
Internationalization		

For more information, see the Developer Magic: RapidApp User's Guide.

xxxvi 007-2006-130

Integrating an Application

This section lists the basic steps for integrating an existing application into the IRIX Interactive Desktop environment. The steps are listed in a very general way, to give you a brief overview of the process.

If you're writing a new application, here are a few tips:

- If possible, use IRIS ViewKit. Refer to the IRIS ViewKit Programmer's Guide for instructions.
- Don't use IRIS GL. Use OpenGL or Open Inventor instead.

Note: Open Inventor isn't part of the IRIS Developer's Option, it is a separate option. In the United States and Canada, call SGI Direct at 800-800-SGI1 (7441) for more information about how to order the Open Inventor Option; outside the United States and Canada, please contact your local sales office or distributor.

To integrate your application into the IRIX Interactive Desktop, follow these steps:

- If your application uses IRIS GL, port to OpenGL if possible. If it's impractical for you to port to OpenGL at this time, at least switch to mixed-model IRIS GL programming, if you haven't already done so. (Mixed-model programs use Xt for event and window management).
 - For information on porting from IRIS GL to OpenGL and for switching your program to mixed-model, refer to the *OpenGL Porting Guide*. This manual is included online in the IRIS Developer's Option (IDO). View it using the IRIS InSight Viewer.
- 2. Set up your application to comply with the IRIX Interactive Desktop look and feel:
 - use the Enhanced IRIS IM look
 - use Schemes
 - use the new and enhanced IRIS IM widgets where appropriate
 - set up your application for correct window, session, and desks management
 - customize the minimize window image for your application (optional)
 - use the extensions provided in the Selection Library and the File Alteration Monitor (optional)

007-2006-130 xxxvii

These topics, as well as information on fonts, are covered in Part 1 of this guide.

- 3. Create Desktop icons for your application and add them to the Icon Catalog. You'll need an icon for the application itself as well as icons for any unusual data formats. See Part 2 of this manual for instructions.
- 4. Use swpkg to package your application so that your users can install it easily. See the *Software Packager User's Guide* for information for instructions on using swpkg.

xxxviii 007-2006-130

Getting the Right Look and Feel

Ι

Chapter 1:

Getting the Right Look and Feel: An Overview

Chapter 2:

Getting the IRIX Interactive Desktop Look

Chapter 3:

Using Schemes

Chapter 4:

Using the Silicon Graphics Enhanced Widgets

Chapter 5:

Window, Session, and Desk Management

Chapter 6:

Customizing Your Application's Minimized Windows

Chapter 7:

Interapplication Data Exchange

Chapter 8:

Monitoring Changes to Files and Directories

Chapter 9:

Providing Online Help With SGIHelp

Chapter 10:

Handling Users' System Preferences

Getting the Right Look and Feel: An Overview

This chapter contains these sections:

- "About the IRIX Interactive Desktop Look and Feel" briefly explains the basics of the IRIX Interactive Desktop look and feel and tells you where to find more detailed information.
- "Getting the Right Look and Feel: The Basic Steps" briefly lists the basic steps for getting the right look and feel and tells you which chapter covers each step.

About the IRIX Interactive Desktop Look and Feel

One of the most important things you can do to integrate your application into the IRIX Interactive Desktop environment is to get the right look and feel. This look and feel is largely based on IRIS IM, the SGI port of the industry-standard OSF/Motif toolkit. In particular, the look and feel is based on an enhanced version of IRIS IM and on the 4Dwm window manager (the SGI mwm-based window manager). The *IRIX Interactive User Interface Guidelines* explains the differences between the IRIX Interactive Desktop look and feel and the OSF/Motif look and feel.

Users have certain expectations of how applications appear and behave in the IRIX Interactive Desktop environment, and by meeting these expectations, you make your application much easier and more pleasant to use. The chapters in this part of the manual explain how to set up your application to provide the IRIX Interactive Desktop look and feel.

Getting the Right Look and Feel: The Basic Steps

To provide the correct look and feel for your application, be sure to:

- 1. **Recompile with libXm** (the IRIS IM version that ships with IRIX 5.3 or later). If your application uses an earlier version of IRIS IM, recompile to make sure that it runs correctly. See the *IRIS IM Release Notes* for information on the differences between the current and previous versions of IRIS IM.
- Use the IRIX Interactive Desktop enhanced appearance. Turn on the IRIX
 Interactive Desktop "look," which enhances the appearance of standard IRIS IM widgets and gadgets. See Chapter 2, "Getting the IRIX Interactive Desktop Look," for instructions.
- 3. **Use schemes.** The schemes mechanism is a simple method for providing user-selectable default colors and fonts for your application. For more information on Schemes, see Chapter 3, "Using Schemes."
- 4. **Use the new and extended widgets (optional).** SGI provides some new IRIS IM widgets, extensions of some existing widgets, and some mixed-model programming widgets (for use with IRIS GL and OpenGL). For more information, see Chapter 4, "Using the SGI Enhanced Widgets."
- 5. Set resources for correct window, session, and desks management. By setting a few important resources, you insure that your application includes the windowing, session management, and desks features that users expect. For instructions, refer to Chapter 5, "Window, Session, and Desk Management."
- 6. Customize minimized icons. SGI provides tools that allow you to easily provide your own look for minimize icons (icons for minimized windows). The tools for creating minimized windows are discussed in Chapter 6, "Customizing Your Application's Minimized Windows."
- 7. **Implement interapplication data exchange.** Interapplication data exchange lets users cut and paste information between your application and other applications. For more information, see Chapter 7, "Interapplication Data Exchange.".
- 8. **Provide online help.** SGI provides an online help system for integrating help with your application. Chapter 9, "Providing Online Help With SGIHelp," describes how to use the online help system.
- 9. **Monitor changes to the filesystem (optional).** SGI provides a File Alteration Monitor (FAM) that your application can use to monitor the filesystem. Chapter 8, "Monitoring Changes to Files and Directories," explains how to use FAM.

Getting the IRIX Interactive Desktop Look

The simplest step in integrating your application with the Desktop environment is to turn on the **IRIX Interactive Desktop "look,"** which enhances the appearance of standard IRIS IM widgets and gadgets. "The IRIX Interactive Desktop Look: Graphic Features and Schemes" in Chapter 3 of the *IRIX Interactive User Interface Guidelines* describes the enhancements.

To turn on the IRIX Interactive Desktop look for an application, simply set the application's **sgiMode** resource to "TRUE." Typically, you should add this line to the /usr/lib/X11/app-defaults file for your application:

appName*sgiMode: TRUE

where appName is the name of your application.

The standard IRIS IM library supports the IRIX Interactive Desktop look. You do not need to link with a separate library or call a special function to enable the IRIX Interactive Desktop look. If you do not turn on the IRIX Interactive Desktop look, your application's widgets and gadgets have the standard IRIS IM appearance.

If your application uses the IRIX Interactive Desktop look, it should also use **schemes**, which are described in Chapter 3, "Using Schemes." SGI designed its color and font schemes to work well with the IRIX Interactive Desktop look.

Using Schemes

Schemes provide an easy way to apply a collection of resources to your application. The scheme mechanism allows your users to select from pre-packaged collections of colors and fonts that are designed to integrate visually with the IRIX Interactive Desktop and other applications. "Schemes for Colors and Fonts" in Chapter 3 in *IRIX Interactive User Interface Guidelines* describes the guidelines for using schemes in the IRIX Interactive Desktop environment.

This chapter contains the following sections:

- "Schemes Overview" on page 7 provides an overview to schemes.
- "Using Schemes in Your Application" on page 9 describes what you need to do to use schemes in your application.
- "Extending a Scheme to Support New Colors" on page 18 provides tips for testing how your application responds to different schemes.
- "Creating New Schemes" on page 21 describes how to create new schemes.
- "Hard-Coding a Scheme for an Application" on page 22 describes how to force your application to use one specific scheme.

Schemes Overview

Schemes allow you to provide default colors and fonts for your application, while also ensuring that users can easily select other color and font collections according to their individual needs and preferences. SGI includes some standard system schemes with the X execution environment, but end users can modify existing schemes or create new ones, and you can create new schemes or extend existing ones for use with your application.

This section provides an overview of schemes and explains why you should use schemes in your application.

Why You Should Use Schemes

As a developer, it is impossible for you to choose colors and fonts for your application that satisfy all users. Aside from the consideration of individual taste, display characteristics vary and some users have various degrees of colorblindness. Schemes allow users to select colors and fonts according to their preferences and needs.

Although users can already use the X resource mechanism to customize colors and fonts, it is very difficult and time-consuming for most end users to do so, because the task requires knowledge of the internal structure of the program. On the other hand, if your application supports schemes, users can use the graphical Schemes Browser, schemebr (available from the "Color Schemes" option of the Customize menu in Desktop toolchest), to change colors and fonts.

Using schemes also reduces the time and effort required to develop your application. Instead of choosing your own colors and fonts and coding them into your application, you can simply set a resource value to activate schemes and get the distinctive IRIX Interactive Desktop appearance.

Basic Scheme Concepts

A scheme simply maps specific colors and fonts to abstract resource names according to the functions they serve in an application. So instead of using specific colors like "blue" or "#123456" and specific fonts like "-*-screen-medium-r-normal--13-*-*--*-iso8859-1," your application can use symbolic values like TextForeground, TextSelectedColor, and FixedWidthFont. The exact definition of these symbolic values depends on the scheme the user chooses to apply to your application. As long as your application uses the symbolic color and font names for the purposes for which they were intended, users or graphic designers can design a new palette (a binding of the symbolic values to specific colors) and the result should look good with your application.

Often, you do not even need to deal with the symbolic colors and fonts yourself. The schemes mechanism includes a map file that automatically binds the symbolic values to the various IRIS IM widgets and widget resources. One case where you may need to set a color or font explicitly is if you need to highlight a component (for example, in a chart). The schemes mechanism defines special symbolic values such as HighlightColor1

through HighlightColor8 for these purposes. (See "Directly Accessing Colors and Fonts" on page 13 for more information on the symbolic values.) Another case where you need to be aware of the symbolic values is if you need to extend a scheme for your application. (See "Extending a Scheme to Support New Colors" on page 18 for more information on extending a scheme.)

Using Schemes in Your Application

This section describes how to write your application for use with schemes and includes:

- "Turning on Schemes for Your Application" on page 9
- "Special Considerations for Programming with Schemes" on page 10
- "Assigning Non-Default Colors and Fonts to Widgets" on page 11
- "Directly Accessing Colors and Fonts" on page 13

Turning on Schemes for Your Application

SGI incorporates schemes in its implementation of Xt, so you do not need to link to a separate schemes library or call a special function to use schemes. All you need to do to enable schemes is to include in your application's app-defaults file (in the /usr/lib/X11/app-defaults directory) the line:

```
AppClass*useSchemes: all
```

AppClass is your application's class name. This activates all aspects of schemes.

Note: To ensure that users do not accidently override your settings, be sure to prefix the **useSchemes** resource with your application's class name.

To deactivate schemes, you can set:

```
AppClass*useSchemes: none
```

If you wish to activate schemes without using an app-defaults file, or if you want to guarantee that the schemes setting cannot be changed by users, call the function SgiUseScheme():

007-2006-130

```
void SgiUseScheme(char *value)
```

value can be either "all" or "none." This function requires that you include the header file <X11/SGIScheme.h>.

For example:

Special Considerations for Programming with Schemes

The schemes map file automatically handles applying colors and fonts to most IRIS IM widgets based on the widgets' class names. Unfortunately, IRIS IM does not have unique class names for menu bars, menu panes, and option menus. To allow schemes to be applied to these elements, your application should use some simple naming conventions for these widgets. Name all menu bars "menuBar," all option menus "optionMenu," and the pane of all option menus "optionPane." Schemes also recognize some other variations of these names, including "menu_bar," "menubar," "menu_Bar," and so on.

If you need to set a color or a font in your application, use the procedures described in "Assigning Non-Default Colors and Fonts to Widgets" on page 11 and "Directly

Accessing Colors and Fonts" on page 13. Do not hard code colors or fonts in your application because they may not work with the scheme that a user selects. For example, if you programmatically set a text color to black and a user chooses a scheme that has a very dark background, your text is unreadable. Also avoid setting colors that IRIS IM normally computes. For example, if you hard code the top or bottom shadow colors used by IRIS IM controls, these colors may not be correct if a user changes the scheme.

There are obviously some cases for which this recommendation does not apply. The most common are windows in which you are rendering images. For example, if your application uses OpenGL or some other library to render an image in a window, the colors used in this window are not derived from schemes.

Fonts are usually less critical than colors, although the best visual effects are produced if you use only the fonts defined in the schemes. You should be aware that on high-resolution screens, the sizes of the fonts defined by schemes can change. Therefore, you should design the layout of your application to handle variable-sized fonts. This means you should not hard-code x, y locations or fixed widths or heights for widgets in your application. Instead use IRIS IM manager widgets such as the Form to achieve a flexible layout that can respond to changes in font sizes.

Assigning Non-Default Colors and Fonts to Widgets

Sometimes, you may want to override the default color or font assigned to a widget by a scheme. For example, all labels are set by default to use a bold font (BoldLabelFont); however you may decide that a regular font (PlainLabelFont) is more appropriate for some of your application's labels.

To assign a non-default font or color to a widget, include a line in your application's app-defaults file mapping a different symbolic scheme resource to that widget. For example, the following line assigns a regular label font (rather than the default bold font) to a label in your application named "simpleLabel":

YourApp*simpleLabel*fontList: SGI_DYNAMIC PlainLabelFont

The symbol SGI_DYNAMIC identifies this resource as a dynamically changeable scheme resource. The actual font assigned to PlainLabelFont can potentially be different in each scheme. As the user changes schemes, the correct resource is applied to your program.

007-2006-130

Note: Remember to prefix the widget hierarchy with your application's class name to prevent users from accidentally overriding your setting.

You can use the same technique with colors. For example, suppose you have two types of label widgets positioned on an IRIS IM XmDrawingArea widget and you want to use color to give some significance to different labels. Perhaps the application is some type of a flowchart and some of the labels represent tasks in progress, while other represent tasks that have been completed. The schemes map file already maps the symbolic scheme resource DrawingAreaColor to the XmDrawingArea widget. The scheme palette also provides colors that both provide a nice contrast against the DrawingAreaColor and allow the current TextForeground color to be readable. These colors are DrawingAreaContrast1, DrawingAreaContrast2, DrawingAreaContrast3, and DrawingAreaContrast4. To specify the colors of each label widget in your application, you can set the following resources:

```
YourApp*label1*background: SGI_DYNAMIC DrawingAreaContrast1
YourApp*label2*background: SGI_DYNAMIC DrawingAreaContrast1
YourApp*label3*background: SGI_DYNAMIC DrawingAreaContrast2...
```

Each scheme also contains a set of basic colors that you can use for simple graphics, icons, and so on. These colors maintain their basic characteristics, but change slightly from scheme to scheme to blend with the general flavor of the scheme. For example, you can set a label widget to be "red" as follows:

```
YourApp*label*background: SGI_DYNAMIC RedColor
```

The exact shade of red changes from scheme to scheme, but always is "reddish" and always fits with the other colors in the scheme.

If necessary, you can also use non-scheme colors and fonts, although SGI strongly recommends that you do not do this. If you hard-code a color, the user may select a scheme in which that color does not provide the contrast you desire. The color can even be "lost" among the other scheme colors. Non-scheme fonts are less likely to cause problems, but your application will have an inconsistent appearance if it uses them.

You use the same methods to assign a non-scheme color or font that you normally would in an X program. For example, you can set a font for a label named "simpleLabel" in your app-defaults file as follows:

```
YourApp*simpleLabel*fontList: 6x12
```

Directly Accessing Colors and Fonts

When your application uses widgets only, the schemes map file automatically retrieves all colors and fonts from the current scheme and assigns them to your application's widgets. However, you may need to access some of the scheme's colors or fonts directly from within a program. For example, you may want to draw a bar chart or other display using colors that look good no matter what scheme the user has selected.

Example 3-1 shows an example of a function that retrieves a color value given a widget, the color resource name, and the color resource class.

Example 3-1 Retrieving a Scheme Color Value

```
Pixel getColorResource(Widget w, char *name, char *classname)
   XtResource request_resources;
   Display *dpy = XtDisplay ( w );
         scr = DefaultScreen ( dpy );
   Colormap cmap = DefaultColormap ( dpy, scr );
   XColor color, ignore;
   char *colorname;
   request_resources.resource_name = (char *) name;
   request_resources.resource_class = (char *) className;
   request_resources.resource_type = XmRString;
   request_resources.resource_size = sizeof (char *);
   request_resources.default_type = XmRImmediate;
   request_resources.resource_offset = 0;
   request_resources.default_addr = (XtPointer) NULL;
   XtGetSubresources(w,
                     (XtPointer) &colorname,
                     NULL, NULL,
                     &requested_resources,
                     1, NULL, 0);
   if ( colorname &&
        XAllocNamedColor ( dpy, cmap, colorname, &color,
                           &ignore ) )
       return ( color.pixel );
   else
       return ( BlackPixel ( dpy, scr ) );
```

007-2006-130

You can then retrieve the color defined by the scheme resource **drawingAreaContrastColor1** using getColorResource() as follows:

where barChartWidget is the widget that you will use the color in.

Tip: There is a simple method for retrieving a resource value if you are using the IRIS ViewKit toolkit. Instead of writing the getColorResource() function listed in Example 3-1, you can call:

You must handle some resources programmatically. For example, the *IRIX Interactive User Interface Guidelines* suggests that your application use a different color for text fields that are not editable than it uses for editable text fields. The IRIS IM text widget currently does not change colors automatically when set to read only mode; so your application must handle this itself. The correct color is provided by schemes as the symbolic name ReadOnlyBackground, and can be retrieved by the resource readOnlyBackground. Assuming that you've created the getColorResource() function listed in Example 3-1, the following code illustrates this process:

Tip: The equivalent IRIS ViewKit code would be:

Pre-Defined Scheme Resources and Symbolic Values

Table 3-1 lists the pre-defined scheme resources and symbolic values. You can use the resources to retrieve color and font values from within your application as described in "Directly Accessing Colors and Fonts" on page 13. You can use the symbolic values to assign colors and fonts to widgets in resource files as explained in "Assigning Non-Default Colors and Fonts to Widgets" on page 11.

 Table 3-1
 Pre-Defined Scheme Resources and Symbolic Values

Resource	Symbolic Value	Intended Use
basicBackground	BasicBackground	Background of application
textForeground	TextForeground	Color of text characters
textBackground	TextBackground	Background of multi-line text widgets
textFieldBackground	TextFieldBackground	Background of single-line text field widgets
readOnlyBackground	ReadOnlyBackground	Background of read-only text and text field widgets
textSelectedBackground	TextSelectedBackground	Background when text is selected with the mouse
textSelectedForeground	TextSelectedForeground	Color of text characters when text is selected with the mouse
disabledTextForeground	DisabledTextForeground	For future use, this color will indicate disabled text instead of stippling.
scrolledListBackground	ScrolledListBackground	Background of scrolled list widgets
scrollBarTroughColor	ScrollBarTroughColor	Trough of scrollbar
scrollBarControlBackground	Scroll Bar Control Background	Scrollbar controls (thumb, searchbutton)
buttonBackground	ButtonBackground	Background of push buttons

 Table 3-1 (continued)
 Pre-Defined Scheme Resources and Symbolic Values

Resource	Symbolic Value	Intended Use	
selectFillColor	SelectFillColor	Fill color for standard IRIS IM radio and toggle buttons	
selectColor	SelectFillColor	IRIS IM toggle and check fill color	
checkColor	CheckColor	IRIX Interactive Desktop toggle check mark color	
radioColor	RadioColor	IRIX Interactive Desktop radio pip color	
indicatorBackground	IndicatorBackground	IRIX Interactive Desktop background color for toggles and radios	
warningColor	WarningColor	Background color for icons in warning dialogs	
errorColor	ErrorColor	Background color for icons in error dialogs	
informationColor	InformationColor	Background color for icons in information dialogs	
wMBackground	WMBackground	Window manager colors. Note	
wMActiveBackground	WMActiveBackground	that 4Dwm currently does not pick up foreground. "Active"	
wMForeground	WMForeground	colors are used for window manager borders with mouse	
wMActiveForeground	WMActiveForeground	focus.	
alternateBackground1	AlternateBackground1	Can be used as background	
alternateBackground2	AlternateBackground2	color for widgets or text areas. Guaranteed to be different from	
alternateBackground3	AlternateBackground3	one another, contrast with basic background and text	
alternateBackground4	AlternateBackground4	background, and can have text drawn on them.	
alternateBackground5	AlternateBackground5	Grawn on them.	
alternateBackground6	AlternateBackground6		

 Table 3-1 (continued)
 Pre-Defined Scheme Resources and Symbolic Values

Resource	Symbolic Value	Intended Use	
drawingAreaBackground	DrawingAreaBackground	Background of drawing area widgets (typically used for graphs)	
drawing Area Contrast Color 1	Drawing Area Contrast Color 1	Contrast colors for drawing	
drawing Area Contrast Color 2	Drawing Area Contrast Color 2	areas (typically used for graphs and trees). These colors are	
drawing Area Contrast Color 3	Drawing Area Contrast Color 3	guaranteed to be different fro one another, different from th	
drawing Area Contrast Color 4	DrawingAreaContrastColor4	drawing area background, and can have text drawn on them	
highlightColor1	HighlightColor1	Bright highlights suitable for	
highlightColor2	HighlightColor2	small color spots. The first four are supposed to be in the same	
highlightColor3	HighlightColor3	hue family as the corresponding Drawing Area Contrast colors so	
highlightColor4	HighlightColor4	that the pair may be used for doing highlights in an annotated	
highlightColor5	HighlightColor5	scrollbar.	
highlightColor6	HighlightColor6	These colors are typically used for outlining and drawing	
highlightColor7	HighlightColor7	graphs, wherever a small	
highlightColor8	HighlightColor8	amount of color needs to be highly visible.	
redColor	RedColor	Colors that can be used for	
orangeColor	OrangeColor	various graphics purposes. These colors always	
yellowColor	YellowColor	approximate their names, but may be slightly adjusted to	
greenColor	GreenColor	blend with each scheme.	
blueColor	BlueColor	Typically used in graphs and charts.	
brownColor	BrownColor		
purpleColor	PurpleColor		
boldLabelFont	BoldLabelFont	Bold labels, such as column headings	

Pre-Defined Scheme Resources and Symbolic Values Symbolic Value Intended Use Resource smallBoldLabelFont SmallBoldLabelFont Labels for tight packing situations Labels where space is at a tinyBoldLabelFont TinyBoldLabelFont premium PlainLabelFont Button labels, also can be used plainLabelFont for values in "Name: Value" pairs smallPlainLabelFont SmallPlainLabelFont Small buttons Menus obliqueLabelFont ObliqueLabelFont Small menus smallObliqueLabelFont SmallObliqueLabelFont fixedWidthFont FixedWidthFont Text areas where fixed width is mandatory, for example where

it's important that columns line

Text where a fixed-width font is appropriate but space is at a

up

premium

Extending a Scheme to Support New Colors

smallFixedWidthFont

Table 3-1 (continued)

SGI strongly recommends that you use existing scheme colors for your application. Using the existing scheme colors ensures that your application will work with all schemes. However, you may need to add colors in some basic cases. This section explains how to extend Schemes to support new colors when necessary.

SmallFixedWidthFont

To extend a scheme, you create new symbolic values for the resources you need and then define bindings between these values and resources that your application can retrieve. This section describes the internal organization of scheme files and then describes how to define these symbolic values and mappings.

18 007-2006-130

Scheme File Organization

All system scheme files are kept in the directory /usr/lib/X11/schemes by default. This directory contains several subdirectories, one for each scheme. The directory Base serves as the basis of all schemes, although it is not a user-identifiable scheme itself. Base contains at least three files: Base, BaseColorPalette, and FontPalette. You may see other files as well.

Each of the palette files (BaseColorPalette and FontPalette) contain cpp-style definitions of symbolic names that represent colors and fonts. The Base file (referred to as a *map* file) contains the default mappings between these symbolic names and specific resources, using the same format as all X resource files.

For example, the Base file contains a line like:

*XmText*foreground: TextForeground

This says that all Text widgets should use the color represented by TextForeground as their foreground color. By default, all schemes share the mappings specified in this Base file. However, the value assigned to each symbolic name can vary from scheme to scheme. For example the Mendocino scheme defines TextForeground to be #fffffe (white) whereas the Milan scheme defines TextForeground to be #000000 (black). The scheme mechanism evaluates these specifications at run time, based on the scheme specified by the user.

By default, all applications pick up the definitions in the Base file and the palette files from the selected scheme. However, applications can load additional scheme files as well.

How to Extend a Scheme

There are two methods for extending a scheme. The first is to add additional mappings between existing colors and widgets in your application. This may be necessary if you want to bind colors of fonts to your widgets in a different way, or if you are using custom widgets that are not handled by the basic scheme mappings. You should do this by adding resources to your application's app-defaults file, using the SGI_DYNAMIC mechanism described earlier.

The second situation occurs when you would like to use colors or fonts that are not provided as a part of the existing schemes. Your should try to avoid this situation when

007-2006-130

at all possible, because users can use the scheme editor to create new schemes, and there is no current mechanism for using the scheme editor to adjust any additional colors your application may define. The standard scheme palettes contain over 40 colors, which should be enough for most situations.

To add new colors to the existing schemes, you need to create a new palette file that contains cpp-style definitions of those colors. To get your application to load this new palette, you then need to set a resource, **paletteFileList**, in your application's app-defaults files that specifies as a comma-separated list all custom palette files to be loaded.

For example, assume you have an application whose class name is Calendar and that you need two colors that you refer to in your program symbolically as **brightColor** and **darkColor**. If no color in the exiting schemes satisfies your requirements for this color, you will need to extend a scheme. Here are the steps you'd follow:

 Define the colors in a palette file. By convention, you should name this file <AppClass>Palette, where <AppClass> is the application class of your application. In this case, you would name the palette CalendarPalette. It would contain the lines:

```
#define BrightColor red
#define DarkColor MidnightBlue
```

You can use any color names available on your target systems, as well as RGB specifications. If the resources you are using are dependent on the gamma setting, you should account for this in the palette file. You can handle the commonly used gamma settings like this:

```
#ifdef GAMMA_1_0
#define BrightColor <color that looks right on 1.0 gamma>
#define DarkColor <color that looks right on 1.0 gamma>
#endif
#ifdef GAMMA_1_7
#define BrightColor <color that looks right on 1.7 gamma>
#define DarkColor <color that looks right on 1.7 gamma>
#endif
#ifdef GAMMA_2_4
#define BrightColor <color that looks right on 2.4 gamma>
#define DarkColor <color that looks right on 2.4 gamma>
#define DarkColor <color that looks right on 2.4 gamma>
#endif
```

The scheme mechanism handles gamma other than those listed here by finding the closest match.

- 2. Create additional palette files as needed. If your new resources need to change values when a scheme changes, create a file named CalendarPalette for each scheme you want to support. For each scheme, choose the colors that look right for that scheme. Note that if you do not support all existing schemes, the values of your new scheme resources defaults to the values in Base; so your application may not look right when a user selects that scheme.
- 3. Set your application's **paletteFileList** resource to load the new palettes. In this example, you would add the following line to your app-defaults file:
 - Calendar*schemeFilePaletteList: CalendarPalette
- 4. When you install your software on a user's machine, you need to install the executable, the app-defaults file, and any other files specific to your application. You must also install your palette files into the directory of each scheme you support. You should, at a minimum, support the schemes found in /usr/lib/X11/schemes. You may also want to support schemes you find installed in /usr/local/schemes.

Testing Your Application with Schemes

For best results, be sure to test your application against all available schemes, and watch for any anomalies. As an added precaution, you can try using the Scheme Browser, schemebr (available from the "Color Schemes" option of the Customize menu in Desktop toolchest), to create some variations on existing schemes and see how your program reacts. If you have not added any resources and are not setting any colors or fonts in your program or app-defaults files, any scheme should be reasonable. If you have extended the schemes or set colors directly in your application, you should watch carefully to see how your application reacts as colors change. It is always possible to use the scheme editor to create a very bad scheme, but if your program seems more sensitive than others to changes, you should think more carefully about your use of color.

Creating New Schemes

You can also include your own new schemes in your software distribution; however, be aware of the following concerns:

 The largest benefit of schemes is the users' ability to change to schemes of their choice; so even if you create a scheme that you prefer for your application, make sure your program looks good with the existing schemes.

• If you install your scheme on a user's system, the user may apply that scheme to other applications. If you attempt to design a new scheme, make sure the scheme works reasonably with other applications on the desktop.

The easiest way to design a new scheme is to use the Scheme Browser, schemebr, available from the "Color Schemes" option of the Customize menu in Desktop toolchest. For best results, you should base your scheme on an existing scheme, preferably one of the standard ones supported by SGI. If you make only minor changes, your new scheme should work with other programs. Once you have created and saved your new scheme, you can retrieve the files from your \$HOME/.desktop-<hostname>/schemes directory, where <hostname> is the name of your system. You can install your scheme in /usr/local/schemes/<SchemeName>, where <SchemeName> is the name you have chosen for your scheme. Once installed, this scheme appears in the Scheme Browser as a local scheme. You can also include this scheme with your software distribution.

Hard-Coding a Scheme for an Application

In some rare situations, you may want your application to use one particular scheme, not the one that the user selects. SGI strongly recommends that you not use this approach, but if your application has special needs, the process is simple to do. Specify the value of the **scheme** resource in your application's app-defaults file using a complete path name. For example:

YourApp*scheme: /usr/lib/X11/schemes/Milan

When using this approach, the location of the scheme directory is unimportant. For example, if you've designed a custom scheme for your application, you can place the scheme directory in special configuration directory for your application. For example:

YourApp*scheme: /usr/lib/YourApp.config/YourAppScheme

This setting implies that /usr/lib/YourApp.config/YourAppScheme is a scheme directory. This means that the directory YourAppScheme would need to contain the files BaseColorPalette (containing the color palette you have defined), a file whose name is the same as the scheme that contains mappings, and a file named FontPalette. Normally, the FontPalette file would be a copy of the file in /usr/lib/X11/schemes/Base, and the map file would be the same as /usr/lib/X11/schemes/Base/Base, but renamed to have the same name as your scheme.

Using the SGI Enhanced Widgets

This chapter discusses the SGI enhanced IRIS IM widgets, the mixed-model programming widgets for using OpenGL in an IRIS IM application, and the HTML viewer widget. This chapter contains these sections:

- "Using the SGI Enhanced Widgets" explains how your application can access the SGI enhanced widgets.
- "Using the Widget Demos" explains how to use the provided demos to experiment with some of the SGI enhanced widgets.
- "The SGI Enhanced Widgets" lists and discusses each of the new widgets.
- "The Enhanced Widgets" lists and discusses each of the enhanced widgets.
- "The Mixed-Model Programming Widgets" discusses the mixed-model programming widgets that SGI provides for use with your OpenGL or IRIS GL application.
- "HTML Viewer Component" explains the HTML viewer component that provides the widgets and libraries necessary for applications to directly access the World Wide Web to display online help and licensing information.

Using the SGI Enhanced Widgets

To use a new or enhanced widget, first switch on the IRIX Interactive Desktop enhanced look and schemes, as described in Chapter 2, "Getting the IRIX Interactive Desktop Look," and Chapter 3, "Using Schemes," respectively.

Using the Widget Demos

SGI provides demos for some of the SGI enhanced widgets. These demos let you experiment with the different resources for each widget.

Location of Widget Demos

The widget demos are in /usr/src/X11/motif/Sgm. The demos are part of the **motif_dev.sw.demoSgi** subsystem—if you cannot find them on your system, check to make sure this subsystem is installed.

Instructions for Building the Widget Demos

The demo tree is shipped with X11 Imakefiles, not Makefiles. To build the demos:

- 1. Change to the IRIS IM demos build tree location.
 - % cd /usr/src/X11/motif/Sgm
- 2. Build the initial Makefile.
 - % ../mmkmf
- 3. Verify that the Makefile is OK.
 - % make Makefile
- 4. Update the rest of your Makefiles.
 - % make Makefiles
- 5. Clean the directory. If you do not not this is not your first installation, obsolete binaries might remain, giving unexpected results.
 - % make clean
- 6. Update Makefile dependencies to make sure that everything is installed properly.
 - % make depend
- 7. Build the demos.
 - % make all

The Enhanced Widgets

SGI provides enhanced versions of these existing IRIS IM widgets:

- "The File Selection Box Widget"
- "The Scale (Percent Done Indicator) Widget"
- "The Text and TextField Widgets"

This section describes how to use the enhancements to these widgets. For guidelines on when to use these widgets, refer to the *IRIX Interactive User Interface Guidelines*.

The File Selection Box Widget

The FileSelectionBox widget (**SgFileSelectionBox**), shown in Figure 4-1, is an enhanced version of the existing IRIS IM FileSelectionBox widget (**XmFileSelectionBox**). The API is consistent with the IRIS IM version of the widget, but the presentation is different.

Note: To get the enhanced FileSelectionBox, you need to set the **SgNuseEnhancedFSB** resource to true (in addition to linking with -1Sgm). Typically, do this in your application's app-defaults file.

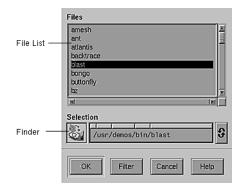


Figure 4-1 The File Selection Box Widget

The FileSelectionBox traverses directories, shows files and subdirectories, and selects files. It has these main areas:

Show menu and Hidden Files toggle

They control what appears in the scrolling list of directories and files beneath them. The Show menu allows the user to restrict the list to display only files of a particular type or format. Minimally the list has two items: All and Custom... "All" always appears at the top of the menu, and "Custom..." always appears at the bottom of the menu following a separator. All shows an unrestricted view of all files and directories in the current subdirectory. Custom... launches the custom filter dialog.

Files and directories list

The scrollable list in the enhanced FileSelectionBox contains both files and directories. Directories appear in bold at the top of the list. Files appear after the directories and are sorted alphabetically.

Finder widget

The DropPocket displays the icon and the text field displays the name of the current directory or file. The user can select a file or directory by dropping its icon on the DropPocket or typing its name in the text field. Automatic file completion is supported. Clicking the right mouse over a path bar button of the path navigation bar shows the directory/file choices at that level. The user can also recall a previously-selected directory from the DynaMenu. "The Finder Widget" on page 42 discusses the Finder widget in more detail.

Command panel

The buttons at the bottom of the FileSelectionBox reflect the action name; the **OK**, **Cancel** and **Help** buttons operate the same in the enhanced FileSelectionBox as they do in the regular version. The **Filter** button pops up a Filter Dialog, which allows a user to enter a shell-style filename expression as filter pattern. The enhanced FileSelectionBox displays only those files in the current directory that match the given pattern. (The FileSelectionBox continues to display any subdirectories in the current directory.)

The programmatic interface to the enhanced FileSelectionBox differs from the regular version in the following points:

- You can retrieve the Finder child of the FileSelectionBox using the standard XmFileSelectionBoxGetChild(3X) by providing the defined constant SgDIALOG_FINDER as the child. You should check the returned widget for validity; it is NULL if the FileSelectionBox is not enhanced.
- XmNdirMask is not guaranteed to be exactly the same as the regular version of the FileSelectionBox in all situations. It does conform to the definition in the XmFileSelectionBox(3X) man page. Specifically, the directory portion XmNdirMask may not be present in the enhanced FileSelectionBox's representation.
- XmNfileTypeMask behavior is different because there is no separate directory list. In the enhanced FileSelectionBox:
 - XmFILE_REGULAR and XmFILE_ANY_TYPE show both files and directories in the file list
 - XmFILE_DIRECTORY shows only directories

For detailed information on the FileSelectionBox widget, refer to the SgFileSelectionBox(3X) man page. For an example program using the FileSelectionBox widget, see "Example Program for File Selection Box" on page 255. See Chapter 10, "Dialogs," in the IRIX Interactive User Interface Guidelines for guidelines on using dialogs in your application. For information about standard XmFileSelectionBox resources, behavior, and callbacks, see the XmFileSelectionBox(3X) man page.

The Scale (Percent Done Indicator) Widget

The Scale widget (**SgScale**), is an enhanced version of the IRIS IM Scale widget (**XmScale**). The enhanced Scale widget is also referred to as the Percent Done Indicator or Progress Indicator.

To implement this indicator, set the following resources:

*scale.sliderVisual: flat_foreground *scale.slidingMode: thermometer

*scale.slanted: true

For an example of code, see "Example Programs for Scale (Percent Done Indicator) Widget" on page 258. Also see "IRIX Interactive Desktop Scales" in Chapter 9 of the *IRIX Interactive User Interface Guidelines* for guidelines on using scales in your application. For more information on the enhanced Scale widget, refer to the SgScale(3X) man page. For more information on the unenhanced version of the widget, refer to the XmScale(3X) man page.

The Text and TextField Widgets

The Text and TextField widgets (**SgText** and **SgTextField**) are enhanced versions of the IRIS IM Text and TextField widgets (**XmText** and **XmTextField**). In addition to the standard **XmText** and **XmTextField** resources, these widgets provide the following new resources:

selectionBackground

The background color for selected text.

selectionForeground

The foreground color for selected text.

errorBackground

The background color for text that you select with an "error status" by using the SgTextSetErrorSelection() or SgTextFieldSetErrorSelection() function (depending on whether the widget is a SgText or SgTextField widget).

cursorVisibleOnFocus

If TRUE (the default), the widget displays the text cursor only when the widget has focus. If FALSE, the cursor is always visible even when the widget does not have keyboard focus.

The SgTextSetErrorSelection() and SgTextFieldSetErrorSelection() functions operate almost identically to the XmTextSetSelection() and XmTextFieldSetSelection() functions. You use them to select a range of text as the primary selection. The only difference is that the selected text is drawn with the background color specified by the **errorBackground** resource instead of that specified by the **selectionBackground** resource.

For a detailed description of the new resources for the enhanced versions of these widgets, refer to the SgText(3X) and SgTextField(3X) man pages. For information on the unenhanced versions of these widgets, refer to the XmText(3X) and

XmTextField(3X) man pages. See "Text Fields" in Chapter 9 of the *IRIX Interactive User Interface Guidelines* for guidelines on using text fields in your application.

The Mixed-Model Programming Widgets

SGI provides two sets of mixed-model programming widgets: one set for use with OpenGL and one set for use with IRIS GL.

A mixed-model program, briefly, is an X program that creates one or more subwindows that use OpenGL or IRIS GL for rendering. Such a program uses Xlib or Xt calls for windowing, event handling, color maps, fonts, and so on. A "pure" IRIS GL application, on the other hand, uses IRIS GL calls for windowing, event handling, color maps, and fonts. (For a more detailed discussion of mixed-model programming, refer to the *OpenGL Porting Guide*.)

If you plan to port your IRIS GL application to OpenGL, a good first step is to port it to mixed-model. The switch to OpenGL is then much easier. The IRIS GL mixed-model widgets make it much easier to port pure IRIS GL applications to mixed-model.

If you are writing a new application, just start with OpenGL and the OpenGL versions of the mixed-model widgets (or use Open Inventor instead of OpenGL—Open Inventor handles all this for you).

The mixed-model widgets are:

IRIS GL OpenGL

GlxDraw GLwDrawingArea GlxMDraw GLwMDrawingArea

The GlxDraw and GLwDrawingArea widgets are suitable for use with any widget set. The GlxMDraw and GLwMDrawingArea widgets are designed especially for use with IRIS IM.

This manual does not tell you how to create a mixed-model program. For instructions on mixed-model programming, refer to the *OpenGL Porting Guide*. (The *OpenGL Porting Guide* contains mixed-model programming information that's relevant for both IRIS GL and OpenGL programmers.)

You can find examples of many mixed-model programs for both OpenGL and IRIS GL in the 4Dgifts directories. If you have trouble finding the relevant directories, refer to the README file in /usr/people/4Dgifts. This README file explains the contents and organization of the 4Dgifts directories.

The SGI Enhanced Widgets

SGI provides these new widgets:

- "The Color Chooser Widget"
- "The Dial Widget"
- "The Drop Pocket Widget"
- "The Finder Widget"
- "The Graph Widget"
- "The Grid Widget"
- "The Springbox Widget"
- "The Thumbwheel Widget"

For guidelines on when to use the different widgets (for example, when to use a Thumbwheel or a Dial) refer to the *IRIX Interactive User Interface Guidelines*.

This section describes each important new IRIS IM widget. It does not discuss new widgets that are part of composite widgets, unless they are generally useful.

The Color Chooser Widget

The ColorChooser widget (**SgColorChooser**) allows users to select colors in RGB or HSV color spaces. Figure 4-2 shows the ColorChooser's default configuration.

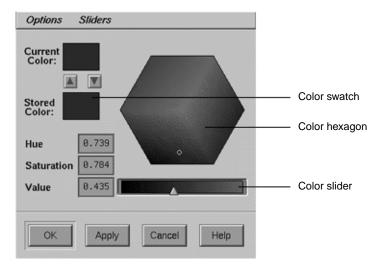


Figure 4-2 The Color Chooser Widget

The ColorChooser includes these components:

- Menus for setting options and sliders for the color chooser.
- A color hexagon that provides visual selection of the hue and saturation components of a color in an HSV color space.
- Color sliders for each of the hue, saturation, value, red, green, and blue color components. To make the color sliders visible, the user can select items from the Sliders menu. (Figure 4-3 shows the ColorChooser with all the sliders visible.) You can also display the color sliders programmatically. Text fields show the exact value of each current color component and allow users to set these values numerically.
- Two color swatches: one for showing the current selected color and one for enabling the user to store a second color for reference.
- Three or four buttons. The default button labels are OK, Cancel, Help, and Apply. If
 the parent of the ColorChooser widget is a DialogShell, then the Apply button is
 managed; otherwise it is unmanaged.

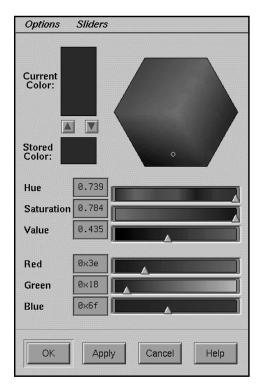


Figure 4-3 The Color Chooser Widget with HSV and RGB Sliders

Users can select a color by manipulating the color hexagon and any of the six sliders, or by changing the values in any of the text fields.

You must include the header file <Sgm/ColorC.h> in any source file that uses a ColorChooser widget.

For more detailed information about the ColorChooser widget, refer to the SgColorChooser (3X) man page. For an example program using the ColorChooser widget, see "Example Program for Color Chooser" on page 242. You can also examine, compile, and experiment with the colorc demonstration program in the directory /usr/src/X11/motif/Sgm/colorc. See "A Specific Standard Support Window: The IRIX Interactive Desktop Color Chooser" in Chapter 6 of the IRIX Interactive Desktop for guidelines on using the ColorChooser widget in your application.

Controlling the Color Chooser Interface

By default, the ColorChooser widget uses GL's Gouraud shading to display the colors in the hexagon and sliders. You can force the ColorChooser widget not to use GL by setting the value of the **SgNuseGl** resource to FALSE. When **SgNuseGl** is FALSE, the ColorChooser widget uses only X function calls. In this case, it does not draw a color hexagon and it uses XmScale widgets instead of the special color sliders.

When using GL, the ColorChooser normally shades the color hexagon and color sliders so that each point is a true representation of the color that would be selected if the user were to move the hexagon pointer or color slider to that point. However, if the value of the **SgNwysiwyg** resource is FALSE, then the ColorChooser always displays the hexagon colors with a Value (intensity) of 1 (maximum intensity), and the RGB sliders with a color range of black to the maximum RGB color component value.

For example, if the current selected color RGB value is (100, 200, 50), then the Red color slider displays the colors (0, 200, 50) through (255, 200, 50) if **SgNwysiwyg** is TRUE, and (0, 0, 0) through (233, 0, 0) if **SgNwysiwyg** is FALSE. (Note that the user can also toggle the value of **SgNwysiwyg** by selecting the "WYSIWYG" option from the ColorChooser's Options menu.)

The **SgNshowSliders** resource determines which of the color sliders are visible. Possible values are:

SgValue Show only the slider for the color Value (the default)

SgRGB_and_Value

Show the Value and RGB sliders

SgRGB_and_HSV

Show all six sliders, the HSV and RGB sliders

The default labels (in the C locale) for the ColorChooser buttons are "OK," "Apply," "Cancel," and "Help." You can change these by setting the values of **SgNokLabelString**, **SgNapplyLabelString**, **SgNcancelLabelString**, and **SgNhelpLabelString** respectively.

You can add additional children to the ColorChooser after creation—they are laid out in the following manner:

• The first child is used as a work area. The work area is placed just below the menu bar

- Buttons—All **XmPushButton** widgets or gadgets, and their subclasses are placed after the **OK** button, in the order of their creation.
- The layout of additional children that are not in the above categories is undefined.

Getting and Setting the Color Chooser's Colors

In ColorChooser callback functions, the RGB color values are provided as the r, g, and b parameters of the SgColorChooserCallbackStruct structure passed to the functions. "Handling User Interaction with the Color Chooser" describes the ColorChooser callbacks.

ColorChooser also provides several convenience routines for getting and setting both the current color values and setting the stored color value.

 ${\tt SgColorChooserSetColor()} \ sets \ both \ the \ current \ and \ the \ stored \ color \ values \ to \ the \ same \ color:$

```
void SgColorChooserSetColor(Widget w, short r, short g, short b);

SgColorChooserGetColor() retrieves the current color values:

void SgColorChooserGetColor(Widget w, short *r, short *g, short *b);

SgColorChooserSetCurrentColor() sets the current color but not the stored color:

void SgColorChooserSetCurrentColor(Widget w, short r, short g, short b);

SgColorChooserSetStoredColor() sets the stored color but not the current color:

void SgColorChooserSetStoredColor(Widget w, short r, short g, short b);
```

For each function, w is the ColorChooser widget and r, g, and b are the red, green, and blue values, respectively.

Handling User Interaction with the Color Chooser

The ColorChooser widget defines the following callback resources:

SgNapplyCallback

Invoked when the user activates the **Apply** button. The callback reason is SgCR_APPLY.

SgNcancelCallback

Invoked when the user activates the **Cancel** button. The callback reason is SgCR_CANCEL.

SgNokCallback

Invoked when the user activates the **OK** button. The callback reason is SgCR_OK.

SgN value Changed Callback

Invoked when the user selects a color. The callback reason is XmCR_VALUE_CHANGED. A color is selected when the user changes the value of a color component with the color hexagon, one of the color sliders, or one of the color components text widgets.

SgNdragCallback

Specifies the list of callbacks called when the user drags the mouse over the color hexagon or one of the color sliders to select a color. The callback reason is XmCR_DRAG.

A pointer to a SgColorChooserCallbackStruct structure is passed to each ColorChooser callback function:

The Dial Widget

The Dial widget (**SgDial**), shown in Figure 4-4, is a new widget that allows users to input or modify a value from within a range of values. Figure 4-4 shows two forms of the Dial widget, one with the input control in the shape of a knob and the other in the shape of a pointer. The user can modify the Dial's value by spinning the knob or pointer. The Dial is usually surrounded by tick marks (marked divisions around the perimeter of the Dial).





Figure 4-4 The Dial Widget in Knob and Pointer Form

You must include the header file <Sgm/Dial.h> in any source file that uses a Dial widget.

For more detailed information about the Dial widget, refer to the SgDial(3X) man page. For an example program using the Dial widget, see "Example Program for Dial" on page 244. You can also examine, compile, and experiment with the <code>dial</code> demonstration program in the directory /usr/src/X11/motif/Sgm/dial. See "Dials" in Chapter 9 of the IRIX Interactive User Interface Guidelines for guidelines on using the Dial widget in an application.

Controlling the Dial Interface

You control the display characteristics of a Dial through widget resources.

The **SgNdialVisual** resource determines whether the Dial uses a knob or a pointer. The default value, SgKNOB, specifies a knob and SgPOINTER specifies a pointer. If you use a pointer, you can also specify the color of the small "indicator" at the center of the pointer using the **SgNindicatorColor** resource; the default color is red.

Specify the position of the lowest value on the Dial with the **SgNstartAngle** resource. The value, which must be between 0 and 360 inclusive, specifies the number of degrees clockwise from the top of the Dial. A default value of 0 corresponds to the top of the Dial.

The **SgNangleRange** resource determines the range of the Dial in degrees. The value, which must be between 0 and 360 inclusive, specifies the number of degrees clockwise

from the start angle of the Dial. The default value of 360 allows the Dial to rotate completely.

The Dial widget displays evenly spaced "tick marks" along the perimeter of the Dial's angle range. You control the number of tick marks with the **SgNdialMarkers** resource; the default number is 16. The length of the tick marks in pixels is determined by the **SgNmarkerLength** resource; the default length is 8 pixels. The **SgNdialForeground** resource determines the color of the tick marks; the default is red.

The resources **XmNminimum** and **XmNmaximum** determine the minimum and maximum values of the Dial. The Dial takes on the minimum value at the position specified by **SgNstartAngle** and takes on the maximum value at the position **SgNangleRange** degrees clockwise from **SgNstartAngle**. The value of **XmNmaximum** must be greater than or equal to the value of **XmNminimum**. The default value of **XmNminimum** is 0 and the default value of **XmNmaximum** is 360.

Getting and Setting the Dial's Value

The **XmNvalue** resource, which must be a value between **XmNminimum** and **XmNmaximum** inclusive, contains the current position of the Dial. You can set or get the value of a Dial widget at any time by respectively setting or getting its **XmNvalue** resource.

In Dial callback functions, the Dial value is provided as the *position* parameter of the SgDialCallbackStruct structure passed to the functions. "Detecting Changes in the Dial's Value" describes the Dial callbacks.

Dial also provides a convenience routine, SgDialSetValue(), for setting the value of **XmNvalue**:

```
void SgDialSetValue(Widget w, int value);
```

w is the Dial widget whose value you want to set and value is the new value.

You can get the current value of a Dial widget at any time by retrieving the value of its **XmNvalue** resource. Dial also provides a convenience routine, SgDialGetValue(), for getting the value of **XmNvalue**:

```
void SgDialGetValue(Widget w, int *value);
```

w is the Dial widget whose value you want to get. Upon returning, *value* contains the Dial's value.

Detecting Changes in the Dial's Value

The Dial widget defines two callback list resources, XmNvalueChangedCallback and XmNdragCallback. A Dial widget invokes XmNvalueChangedCallback whenever its value changes either programmatically (for example, by calling SgDialSetValue()) or through user interaction. A Dial widget invokes XmNdragCallback whenever the user clicks and drags, or "spins," the Dial's knob or pointer.

A pointer to a SgDialCallbackStruct structure is passed to each Dial callback function:

```
typedef struct {
    int reason;
    XEvent *event;
    int position;
} SgDialCallbackStruct;
```

The SgDialCallbackStruct parameters are:

reason The reason the callback was invoked. This value is

XmCR_VALUE_CHANGED in the event of a

XmNvalueChangedCallback and XmCR_DRAG in the event of a

XmNdragCallback.

event A pointer to the XEvent that triggered the callback

position The new Dial value

The Thumbwheel Widget

The ThumbWheel widget (**SgThumbWheel**), shown in Figure 4-5, is a new widget that allows users to input or modify a value, either from within a range of values or from an unbounded (infinite) range.

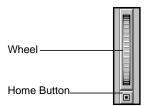


Figure 4-5 The Thumbwheel Widget

A ThumbWheel has an elongated rectangular region within which a wheel graphic is displayed. Users can modify the ThumbWheel's value by spinning the wheel. A ThumbWheel can also include a home button, located outside the wheel region. This button allows users to set the ThumbWheel's value to a known position.

You must include the header file <Sgm/ThumbWheel.h> in any source file that uses a Thumbwheel widget.

For detailed information on the ThumbWheel widget, refer to the SgThumbWheel (3X) man page. For an example program using the ThumbWheel widget, see "Example Program for ThumbWheel" on page 252. You can also examine, compile, and experiment with the thumbwheel demonstration program in the directory /usr/src/X11/motif/Sgm/thumbwheel. See "Thumbwheels" in Chapter 9 of the IRIX Interactive User Interface Guidelines for guidelines on using the ThumbWheel widget

Controlling the ThumbWheel Interface

in your application.

You control the display characteristics of a ThumbWheel through widget resources.

The resources **XmNminimum** and **XmNmaximum** determine the minimum and maximum values of the ThumbWheel. Setting **XmNmaximum** equal to **XmNminimum** indicates an infinite range.

The default value of **XmNminimum** is 0 and the default value of **XmNmaximum** is 100.

The **SgNangleRange** resource specifies the angular range, in degrees, through which the ThumbWheel is allowed to rotate. The default of 150 represents roughly the visible amount of the wheel. Thus clicking at one end of the wheel and dragging the mouse to the other end would give roughly the entire range from **XmNminimum** to **XmNmaximum**.

In conjunction with **XmNmaximum** and **XmNminimum**, the **SgNangleRange** resource controls the fineness or coarseness of the wheel control when it is not infinite. If this value is 0, the ThumbWheel has an infinite range. If the range of the ThumbWheel is infinite, you can use the **SgNunitsPerRotation** resource to specify the change in the ThumbWheel's value for each full rotation of the wheel.

If the value of **SgNshowHomeButton** is TRUE, the default, the ThumbWheel displays a home button by the slider. The user can click on the home button to set the value of the

007-2006-130

ThumbWheel to a known value, which is specified by the **SgNhomePosition** resource. The default value of **SgNhomePosition** is 50.

The **XmNorientation** resource determines whether the orientation of the ThumbWheel is vertical, indicated by a value of XmVERTICAL, or horizontal, indicated by a value of XmHORIZONTAL. The default value is XmVERTICAL.

Getting and Setting the ThumbWheel's Value

The **XmNvalue** resource contains the current position of the ThumbWheel. **XmNvalue** must be a value between **XmNminimum** and **XmNmaximum** if the ThumbWheel is not "infinite." You can set or get the value of a ThumbWheel widget at any time by respectively setting or getting its **XmNvalue** resource.

In ThumbWheel callback functions, the ThumbWheel value is provided as the *value* parameter of the SgThumbWheelCallbackStruct structure passed to the functions. "Detecting Changes in the ThumbWheel's Value" describes the ThumbWheel callbacks.

Detecting Changes in the ThumbWheel's Value

The ThumbWheel widget defines two callback list resources,

XmNvalueChangedCallback and XmNdragCallback. A ThumbWheel widget invokes XmNvalueChangedCallback whenever its value changes either programmatically (that is, by setting the value of XmNvalue) or through user interaction. A ThumbWheel widget invokes XmNdragCallback whenever the user clicks and drags, or "spins," the ThumbWheel's wheel.

A pointer to a SgThumbWheelCallbackStruct structure is passed to each ThumbWheel callback function:

The SgThumbWheelCallbackStruct parameters are:

reason The reason the callback was invoked. This value is

XmCR_VALUE_CHANGED in the event of a

XmNvalueChangedCallback and XmCR_DRAG in the event of a **XmNdragCallback**.

event A pointer to the XEvent that triggered the callback.

position The new ThumbWheel value.

The Drop Pocket Widget

The Drop Pocket widget (**SgDropPocket**), shown in Figure 4-6, receives and displays Desktop icons.



Figure 4-6 The Drop Pocket Widget (on left) As Part of the Finder Widget

When users drop Desktop file icons onto the Drop Pocket, the Drop Pocket determines the name of the icon and returns information describing the icon to the application in the callback.

When users drag an acceptable icon over the Drop Pocket, the Drop Pocket background changes color and the Drop Pocket displays the dropped icon. If the type of a file is not known, or if the file does not exist (for example, if the user is specifying a new file), the Drop Pocket displays the icon for unknown file types (this icon looks like a round balloon).

For more information on the Drop Pocket widget, refer to the SgDropPocket (3X) man page. You can find example code in /usr/src/X11/motif/Sgm/dropPocket directory. For an example program using the Drop Pocket widget, see "Example Program for Drop Pocket" on page 247.

man Page Text

SYNOPSIS #include <Sgm/DropPocket.h>

New Resources

SgNDropPocketActivePixel

Specifies the color for the background of the DropPocket when an icon that can be dropped is above the DropPocket.

SgNiconDataBasePath

Specifies the location of the icon database. The default value is /usr/lib/filetype/workspace.otr. Setting this resource to a filename that is not a legal icon database will cause serious problems for the DropPocket.

SgNname

Specifies the compound string that is the name of the current icon. By setting this resource, the application can control the initial icon that appears in the DropPocket. If this resource is NULL, the DropPocket will appear empty. Refer to XmString(3X) for more information on the creation and structure of compound strings.

Callback Information

The Callback structure is SgDropPocketCallbackStruct. A pointer to the following structure is passed to each callback:

reason Indicates why the callback was invoked. The constant

SgCR_ICON_CHANGE is the reason associated with callbacks

generated by a successful icon drop on the DropPocket.

event Points to the XEvent that triggered the callback

window Specifies the window of the DropPocket.

iconName Specifies the name of the icon. For icons representing files, this is the file

name. For other types of icons, the name may not completely specify the icon. For example a person icon may have the name jake, but the icon

represents a person in /usr/lib/faces/jake.

iconData The full string description of the icon.

The Finder Widget

The Finder widget (**SgFinder**), shown in Figure 4-7, is a new widget that accelerates text selection of long objects such as filenames. (A good way to experiment with a Finder widget is to select "An Icon" from the Find toolchest.)

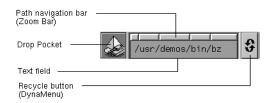


Figure 4-7 The Finder Widget

The Finder widget is customizable for various applications (it is not just for looking at directories; see the SgFinder(3X) man page for customization details). The Finder widget includes four components:

Text field Displays the name of a file or directory.

Path navigation bar

Contains buttons representing each directory in the pathname. When the user clicks on a path bar button, the Finder sets the current directory to the directory listed underneath that button. The path bar is created with an **SgZoomBar**(3X) widget.

Recycle button When users click on the Recycle button, the recycle list appears listing the directories that the user has selected during the current Finder session. Selecting an item from the recycle list changes the current directory to the selected directory. The recycle button is created with an SgDynaMenu(3X) widget.

Drop pocket

Displays the Desktop file icon for the file listed in the text field. The user can drop Desktop file icons into the drop pocket to find the pathname for the file and drag icons out of the drop pocket and put them on the Desktop. The recycle button is created with an **SgDropPocket**(3X) widget.

You must include the header file <Sgm/Finder.h> in any source file that uses a Finder widget.

For more detailed information on the Finder widget, refer to the SgFinder(3X), SgDropPocket(3X), and SgDynaMenu(3X) man pages. For an example using the Finder widget, see "Example Program for Finder" on page 250. You can also examine, compile, and experiment with the finderTest demonstration program in the directory /usr/src/X11/motif/Sgm/finder. See "File Finder" in Chapter 9 of the *IRIX*

007-2006-130 43 *Interactive User Interface Guidelines* for guidelines on using the Finder widget in your application.

Controlling the Finder Interface

If you do not need the drop pocket feature of the Finder widget, you can set the value of the resource **SgNuseDropPocket** to FALSE when you create the widget. This bypasses the costs of setting up drag and drop and loading the file icon libraries. Note that you cannot set this resource using XtSetValues(); if you do not originally create a Finder widget with a drop pocket, you cannot add one afterwards.

Similarly, if you do not need the Recycle button, you can set the value of the resource **SgNuseHistoryMenu** to FALSE. Note that you cannot set this resource using XtSetValues(); if you do not originally create a Finder widget with a Recycle button, you cannot add one afterwards.

You can customize the appearance of the Recycle button by setting the value of the **SgNhistoryPixmap** resource to the pixmap you want to display.

By default, the Finder widget determines where to place the buttons on the path navigation bar by the location of the forward slash (/) character in the text field. You can specify a different separator character by providing it as the value of the **SgNseparator** resource. This feature is useful if you want to use the Finder widget to display something other than filenames.

Getting and Setting Finder Values

```
You can retrieve the current value of the Finder's text field with SgFinderGetTextString(): char *SgFinderGetTextString(Widget w);

You can set the value of the text field with SgFinderSetTextString(): void SgFinderSetTextString(Widget w, char *value);

You can add an item to the "history list" of the Recycle button with SgFinderAddHistoryItem(): void SgFinderAddHistoryItem(Widget w, char *str);
```

You can clear the Recycle button's history list with SgFinderClearHistory():

```
void SgFinderClearHistory(Widget w);
```

You can access a widget component within a finder using SgFinderGetChild():

Widget SgFinderGetChild(Widget w, int child);

child specifies the component and can take any of the following values:

SgFINDER_DROP_POCKET

The drop pocket

SgFINDER_TEXT

The text field

SgFINDER_ZOOM_BAR

The path navigation bar

SgFINDER_HISTORY_MENUBAR

The Recycle button

Handling User Interaction with the Finder

When the user clicks a button in the path navigation bar, the default action of the Finder is to set the current directory to the directory listed underneath that button. You can change this behavior by setting the **SgNsetTextSectionFunc** resource to the handler you want to use. The handler function must be of type SgSetTextFunc, which is defined in <Sg/Finder.h>:

```
typedef void (*SgSetTextFunc)(Widget finder, int section);
```

The first argument is the Finder widget and the second is an integer corresponding to the button pressed. Buttons are numbered sequentially from the left, starting with 0. You can perform whatever operations you want in this function, but typically you include a call to SgFinderSetTextString() to set the value of the text field after the user clicks a button.

Additionally, the Finder widget defines two callback list resources:

XmNactivateCallback

Invoked when the user clicks a path navigation bar button, when the text field generates an activateCallback (for example, the user presses the <Return> key in the text field), or when you set the text string by calling SgFinderSetTextString(). A pointer to an XmAnyCallbackStruct structure is passed to each callback function. The reason sent by the callback is XmCR_ACTIVATE.

XmNvalue Changed Callback

Invoked when text is deleted from or inserted into the text field. A pointer to an XmAnyCallbackStruct structure is passed to each callback function. The reason sent by the callback is XmCR_VALUE_CHANGED.

The Graph Widget

The Graph widget (**SgGraph**) allows you to display any group of widgets as a graph, with each widget representing a node. Figure 4-8 shows an example of a Graph widget.

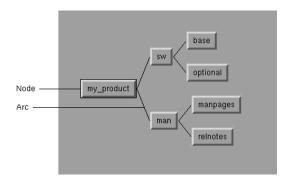


Figure 4-8 The Graph Widget

The arcs used to connect the nodes are instances of an Arc widget (**SgArc**), developed specifically for use with the Graph widget.

The Graph widget allows you to display any group of widgets as a graph, with each widget representing a node. The graph can be disconnected and can contain cycles. The arcs used to connect the nodes are instances of an Arc widget (**SgArc**), developed specifically for use with the Graph widget. Arcs may be undirected, directed, or bidirected. Note that the Graph widget does not understand the semantics of arc direction; in other words, for layout and editing purposes, an Arc will always have a parent and a child regardless of its direction.

The Graph widget has the ability to arrange all nodes either horizontally or vertically according to an internal layout algorithm, and supports an edit mode in which arcs and nodes may be interactively repositioned as well as created. There is also a read-only

mode in which all events are passed directly to the children of the Graph widget. In edit mode, the Graph takes over all device events for editing commands.

The Graph is a complex widget, and a full discuss of its resources, utility functions, and capabilities is beyond the scope of this document. For detailed information about the Graph and Arc widgets, refer to the SgGraph(3X) and SgArc(3X) man pages.

You must include the header file <Sgm/Graph.h> in any source file that uses a Graph widget. You must include the header files <Sgm/Graph.h> and <Sgm/Arc.h> in any source file that uses an Arc widget.

The Springbox Widget

The SpringBox widget (**SgSpringBox**) is a new container widget that arranges its children in a single row or column based on a set of spring constraints assigned to each child. You can use the SpringBox widget to create layouts similar to those supported by the **XmForm** widget, but the SpringBox widget is usually easier to set up.

The value of the SpringBox widget's XmNorientation resource determines its orientation. The default value, XmHORIZONTAL, specifies a horizontal SpringBox and the value XmVERTICAL specifies a vertical SpringBox.

To use the SpringBox, you set constraint resources on each child of the widget to specify the "springiness" for both the widget's size and position relative to its siblings.

You control the springiness of a widget's size by setting the values of its XmNverticalSpring and XmNhorizontalSpring resources. A value of zero means the child cannot be resized in that direction. For non-zero values, the values are compared to the values of other springs in the overall system to determine the proportional effects of any resizing. For example, a widget with a springiness of 200 would stretch twice as much as a widget with a springiness of 100. The default value of both resources is zero.

The values of the resources XmNleftSpring, XmNrightSpring, XmNtopSpring, and XmNbottomSpring control the springiness of a widget's position in relation to its neighboring boundaries. By default, the value of each of these springs is 50. A value of zero means that the SpringBox widget cannot add additional space adjacent to that part of a widget. Larger values are considered in relation to all other spring values in the system.

You must include the header file <Sgm/SpringBox.h> in any source file that uses a SpringBox widget. For more detailed information on the SpringBox widget, refer to the SgSpringBox(3X) man page.

The Grid Widget

The Grid widget (**SgGrid**) is a new container widget that arranges its children in a two-dimensional grid of arbitrary size. You can separately designate each row and column of the grid as having a fixed size or as having some degree of stretchability. You can also resize each child in either or both directions, or force a child to a fixed size.

You must include the header file <Sgm/Grid.h> in any source file that uses a Grid widget. For detailed information on the Grid widget, refer to the SgGrid(3X) man page.

Setting Grid Characteristics

You specify the number of rows and columns in a Grid by setting the values of its **XmNnumRows** and **XmNnumColumns** resources, respectively. The default value for each is 1. Note that you can set the size of a Grid only when you create it; you cannot use XtSetValues() to change the number of rows or columns in a Grid.

The **XmNautoLayout** resource determines the layout policy for a Grid. If its value is TRUE (the default), all rows and columns that have a non-zero resizability factor (described below) are sized according to the desired natural size of the widgets in that row or column.

If **XmNautoLayout** is FALSE, all widgets in resizable rows or columns are sized according to the resizability factor for that row or column. By default, the resizability factor is "1" for all rows and columns, which results in each cell in the grid having an equal size. You can change the resizability factor for a row or column by calling SgGridSetRowMargin() or SgGridSetColumnMargin() respectively:

widget is the Grid widget. The second argument specifies the row or column. Rows are numbered sequentially from the top starting at 0; columns are numbered sequentially

from the left starting with 0. *factor* is the resizability factor for the row or column. Setting this value to 0 establishes the specified row or column as not resizable, regardless of the setting of **XmNautoLayout**. Other values are taken relative to all other rows. For example, if a Grid has three rows whose resizability factors are set to 100, 100, and 200, the first and second rows will occupy one quarter of the space (100/(100+100+200)), while the third row will occupy one half of the available space.

The XmNdefaultSpacing resource default spacing between rows and columns. The default value is 4 pixels. You can override the value on a per row/column basis using SgGridSetColumnMargin() or SgGridSetRowMargin() respectively:

widget is the Grid widget. The second argument specifies the row or column. margin specifies the margin in pixels between the row or column's edges and the widgets it contains. The margin is added to both sides of each row or column, so adding a 1 pixel margin increases the relevant dimension of the affected row or column by 2 pixels.

You can display the boundaries of a Grid by setting the value of its **XmNshowGrid** resource to TRUE. You might find this useful for debugging resize specifications. The default value is FALSE.

Setting Constraints on the Child Widget of a Grid

The **XmNrow** and **XmNcolumn** resources of a Grid's child widget specify the row and column in which the Grid places the child. If you do not specify these values, the Grid widget places the child in a randomly selected unoccupied cell.

The **XmNresizeVertical** and **XmNresizeHorizontal** resources determine whether the Grid can resize the child to fill the cell in the vertical and horizontal directions. The default value of TRUE allows the Grid to resize the child.

If a child is a fixed size, and smaller than the cell that contains it, the child's position within the cell is determined by an **XmNgravity** resource. Gravity may be any of the gravity values defined by Xlib except StaticGravity and ForgetGravity. The default is NorthWestGravity. Note that gravity has no effect if both **XmNresizeVertical** and **XmNresizeHorizontal** are TRUE.

Examples of Using the Grid Widget

Example 4-1 creates a grid of four buttons that all size (and resize) equally to fill one quarter of their parent.

Example 4-1 An Example of Using the Grid Widget

```
createGrid(Widget parent)
    int n;
    Arg args[10];
    Widget grid, child1, child2, child3, child4;
    XtSetArg(args[n], XmNnumRows, 2); n++;
    XtSetArg(args[n], XmNnumColumns, 2); n++;
    grid = SgCreateGrid(parent, "grid", args, n);
    child1 = XtVaCreateManagedWidget("child1",
                                      xmPushButtonWidgetClass,
                                      grid,
                                      XmNrow, 0,
                                      XmNcolumn, 0,
                                      NULL);
    child2 = XtVaCreateManagedWidget("child2",
                                      xmPushButtonWidgetClass,
                                      grid,
                                      XmNrow, 0,
                                      XmNcolumn, 1,
                                      NULL);
    child3 = XtVaCreateManagedWidget("child3",
                                      xmPushButtonWidgetClass
                                      grid,
                                      XmNrow, 1,
                                      XmNcolumn, 0,
                                      NULL);
    child4 = XtVaCreateManagedWidget("child4",
                                      xmPushButtonWidgetClass
                                      grid,
                                      XmNrow, 1,
                                      XmNcolumn, 1,
                                      NULL);
    XtManageChild(grid);
}
```

Example 4-2 creates four buttons. The top row has a fixed vertical size, while the bottom row is resizable. The left column has a fixed size, but the right column can be resized. The

button in the lower right can be resized, but the others cannot. The button in the lower left cell, which can be resized vertically, floats in the middle of its cell. The button in the upper right stays to the left of its cell.

Example 4-2 Another Example of Using the Grid Widget

```
createGrid(Widget parent) {
    int n;
   Arg args[10];
    Widget grid, chidl1, child2, child3, child4;
   n = 0;
   XtSetArg(args[n], XmNnumRows, 2); n++;
    XtSetArg(args[n], XmNnumColumns, 2); n++;
    grid = SgCreateGrid( parent, "grid", args, n );
    SgGridSetColumnResizability(grid, 0, 0);
    SgGridSetRowResizability(grid, 0, 0);
    child1 = XtVaCreateManagedWidget("child1",
                                 xmPushButtonWidgetClass,
                                 grid,
                                 XmNrow, 0,
                                 XmNcolumn, 0,
                                 NULL);
    child2 = XtVaCreateManagedWidget("child2",
                                 xmPushButtonWidgetClass,
                                 grid,
                                 XmNrow, 0,
                                 XmNcolumn, 1,
                                 XmNresizeHorizontal, FALSE,
                                 XmNgravity, WestGravity,
                                 NULL);
    child3 = XtVaCreateManagedWidget("child3",
                                 xmPushButtonWidgetClass,
                                 grid,
                                 XmNrow, 1,
                                 XmNcolumn, 0,
                                 XmNresizeVertical, FALSE,
                                 XmNgravity, CenterGravity,
                                 NULL);
    child4 = XtVaCreateManagedWidget("child4",
                                 xmPushButtonWidgetClass,
                                 grid,
                                 XmNrow, 1,
```

```
XmNcolumn, 1,
NULL);
XtManageChild(grid);
}
```

HTML Viewer Component

The HTML viewer component provides the widgets and libraries necessary for applications to include direct Web access for information from within an application (no external applications have to be run). Thus, applications can provide direct Web access to licensing, online help, and information retrieval.

For example, applications can use this component with a form and the post method to obtain a license for a user on a local or remote server. Also, applications can use the component to format text and images as read-only text, or provide help to the user. Figure 4-9 shows an example of the HTML widget.

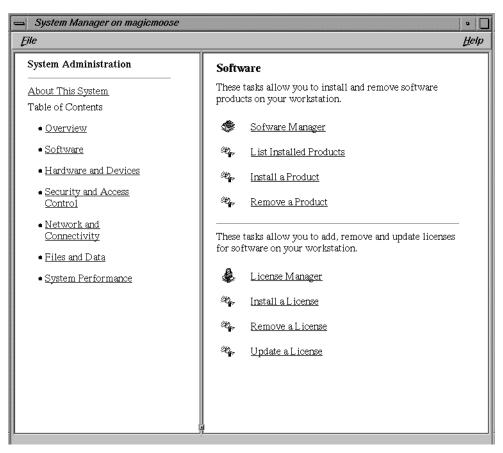


Figure 4-9 The HTML Widget

This section covers the following aspects of the HTML viewer component:

- "Overview of the HTML Viewer Components" on page 54
- "Viewer Components" on page 54
- "Supported Tags and Attributes" on page 55

Overview of the HTML Viewer Components

The HTML viewer makes the World Wide Web accessible from any application. Prior to the Web, applications had to develop a protocol for remote help and construct a server to answer the protocol. Today, however, the Web supplies a predesigned protocol. And the HTML viewer provides the additional components to implement a Web viewer.

The HTML viewer understands how to render HTML markup language. The viewer is derived from the widget and class library licensed for the SGI WebMagic authoring environment. The library allows each application to decide exactly what it needs to support. This component is not meant to supplant general Web browsers which have support for things like bookmarks and search engines. However, the HTML viewer component answers the need for online access to licensing and help.

Viewer Components

The viewer ViewKit component uses the Motif widget to access Web sites, a prebuilt GUI, and provides browsing, navigation, and history control. The ViewKit component requires the use of C++. It is composed of the following components:

- VkWebViewerBase class, which is for programmers who want to provide their own interface. VkWebViewerBase is, from a widget point of view, just a scrolled window with a viewer in it. The scrolled window is held in a form. The subclass uses this form to add the other controls.
- VkWebViewer class, which is a more complete browser for an application programmer that does not want to do a lot of programming with the component.

The include file is /usr/include/Vk/VkWebViewer.h. For example, to create the widget:

```
_viewer = new VkWebViewer("Viewer", mainWindowWidget());
addView(_viewer);
```

For more information about ViewKit, see the IRIS ViewKit Programmer's Guide.

Supported Tags and Attributes

Table 4-1 lists the HTML viewer supported tags. The tags are basically all of HTML 2, selective tags of HTML 3, and extended tags from Netscape and SGI.

 Table 4-1
 HTML Viewer Tags and Attributes

Tag	Attributes		
A	HREF, TARGET, NAME		
AREA	HREF, TARGET, COORDS, SHAPE		
BASE	HREF, TARGET		
BASEFONT	SIZE (17, or exact size such as 24pt)		
BLOCKQUOTE, BQ			
BR			
BODY	BACKGROUND, VLINK, LINK. ALINK, BGCOLOR, TEXT, LANG		
CAPTION	ALIGN		
DIV	ALIGN		
DIR, OL, MENU, UL	START (for OL). See also P, Hn		
DL			
DT			
FRAME	SRC, NAME, WIDTH, HEIGHT		
FRAMESET	COLS, ROWS		
FORM	METHOD, ACTION, TARGET		
HEAD			
HTML			
HR	SIZE, WIDTH, NOSHADE, ALIGN		
IMG	SRC, NOFLOW, ALT, ALIGN, WIDTH, HEIGHT, BORDER, VSPACE, CLEAR		
INPUT	NAME, VALUE, TYPE, SIZE, MAXLENGTH, CHECKED		

Table 4-1 (continued)	HTML Viewer Tags and Attributes			
Tag	Attributes			
ISINDEX	PROMPT			
LI				
LISTING	Deprecated			
MAP	NAME			
NOBR				
NOFRAMES				
OPTION	VALUE, SELECTED			
P, Hn	CLEAR			
PLAINTEXT	Deprecated			
PRE				
SELECT	NAME, SIZE, MULTIPLE			
TABLE	CELLSPACING, CELLPADDING, BORDER, ALIGN, WIDTH, NOFLOW, CLEAR, BGCOLOR			
TD, TH	ROWSPAN, COLSPAN, ALIGN, VALIGN, BGCOLOR, NOWRAP			
TEXTAREA	NAME, ROWS, COLS			
TITLE				
TR	ALIGN, VALIGN, BGCOLOR			
WBR				
XMP	Deprecated			

Table 4-2 lists HTML viewer character tags.

 Table 4-2
 HTML Viewer Character Tags

ADDRESS	В	BIG				
CITE	CODE	DFN				
EM	I	KBD				
FONT SIZE=(07)						
COLOR="#rrbbgg" a						
S	SAMP	SMALL				
STRIKE	STRONG	SUB				
SUP	IT	U				
UL	VAR					

a. COLOR="#rrbbgg" or COLOR="colorname" $\,$

Window, Session, and Desk Management

This chapter contains these sections:

- "Window, Session, and Desk Management Overview" on page 59 briefly discusses window, session, and desk management on Silicon Graphics systems.
- "Implementing an Application Model" on page 65 describes how to structure your application to follow one of the four application models.
- "Interacting With the Window and Session Manager" on page 68 describes how to create windows and interact with the window and session manager.

Window, Session, and Desk Management Overview

This section briefly discusses features of window, session, and desk management on Silicon Graphics system:

- "Window Management"
- "Session Management"
- "Desk Management"

This section also provides a list of references for further reading on window and session management.

Window Management

4Dwm, which is based on mwm (the Motif Window Manager), is the window manager typically used on Silicon Graphics workstations. It provides functions that allow both users and programmers to control elements of window states such as: placement, size, icon/normal display, and input-focus ownership. In addition to window management, 4Dwm provides session and desks management.

Chapter 3, "Windows in the IRIX Interactive Desktop Environment," of the *IRIX Interactive User Interface Guidelines* discusses the interactions and behaviors that your application's windows should support. "Interacting With the Window and Session Manager" on page 68 describes how to comply with the style guidelines.

See IRIS Essentials for more information about the features 4Dwm provides for your users. See the mwm(1X) and 4Dwm(1X) reference pages for more information about the features 4Dwm provides.

Session Management

This section describes session management and explains how to add it to your application. Topics include:

- "Overview of Session Management" on page 60
- "Adding Session Management to Your Application" on page 62
- "Setting the WM_COMMAND String" on page 62
- "Saving Session Information to a File" on page 62

Overview of Session Management

Session management allows users to log out, and any applications that are running at logout automatically restart when they log back in. The 4Dwm window manager keeps a list of the applications and desks that were previously running when the user last logged out and restarts them when the user logs in again.

For your application to be restarted via the 4Dwm session manager, the application must register its initial state with the session manager and make sure the current state is registered at all times.

Additionally, your application should restart in the same state it was in when the user logged out (for example, the same windows open, the same files open, and so on). To support this, you need to design your application so that when the 4Dwm session manager restarts it, it can redisplay any of its co-primary or support windows that were open when the user logged out, reopen any data files that were open, and so on. You can support this either by providing command-line options to your application or other mechanisms such as a state file that your application reads when it is launched.

Types of Session Management: Continuous and Explicit

The two types of session management include **continuous** session management and **explicit** session management. Continuous session management restarts the applications that were running when the user last logged out of the window manager. This is the default setting.

Explicit session management ignores the windows that were open when the user last logged out and always opens a particular set of windows that the user has chosen. Users can configure the windows on the desktop by using the Window Settings Control Panel. They can launch this panel via the Toolchest. From the Toolchest, open the Desktop menu, select "Customize," and then select "Windows." The Windows Settings dialog box appears (see Figure 5-1). The "Save Windows & Desks" item on the Window Settings Control Panel configures either continuous or explicit session management. A user can select explicit, and then press the "Set Home Session" button to save the (current) explicit window configuration. Also, users can launch this control panel from the Icon Catalog's Control Panel page.

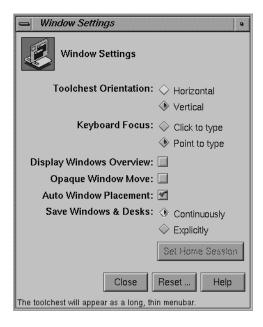


Figure 5-1 Window Settings Control Panel

Adding Session Management to Your Application

Applications can communicate with the window manager by setting properties on the top level window. The WM_COMMAND property gives the window manager the command line that can be used to re-invoke the application in its current state. The 4Dwm window manager sends a WM_SAVE_YOURSELF message to each window that subscribes to tell it to update its WM_COMMAND property and then reads in the value. If the user selects continuous session management, 4Dwm sends the message every 10 minutes and at logout. If the user selects explicit session management, the window manager only queries the applications when the user presses the "Set Home Session" button.

Setting the WM_COMMAND String

If you use ViewKit or XtAppInitialize(3Xt), the initial WM_COMMAND string is set for you when the top level window is realized. Use the xprop(1) command to make sure the WM_COMMAND string is set correctly for the top level window. For example:

```
WM_COMMAND(STRING) = { "webmagic", "/usr/tmp/sgiLook.html" }
```

Even if WM_COMMAND is initially set by your toolkit, you need to keep WM_COMMAND updated if your program changes its state. For instance, if the user renames a data file or successfully opens a new data file, you need to change the WM_COMMAND string with the XSetCommand(3X11):

```
XSetCommand(Display *display, Window w, char **argv, int argc);
```

This can be done in the function that changes the state, and is simpler than responding to the window manager's WM_SAVE_YOURSELF message.

Saving Session Information to a File

If your application already saves state information to a file instead of using the command line, this "state file" can also be used for session management. To work correctly with the user's Windows Control Panel setting, the application should update the file only in response to the window manager's WM_SAVE_YOURSELF message (see Example 5-1, saveyourself.c). This strategy does not work correctly if several instances of your application are able to run at the same time. Only applications that enforced a "Run Once" policy can rely on this strategy.

For more information, see the ViewKit reference page, VkRunOnce(3x), and XSetCommand(3X11).

Debugging Tips

When debugging:

- Use xprop(1) to see the WM_COMMAND string property.
- Make sure that you use the full pathname for data file arguments, which typically are not referenced in the user's path.
- If your application has multiple windows, only set WM_COMMAND for the top-level window.

Example 5-1 Session Management Example Code: saveyourself.c

```
* /
/* saveyourself.c
                                                          * /
/* Example code for handling the window manager's
                                                          * /
/* WM SAVE YOURSELF Protocol
                                                          * /
/*
                                                          * /
/* cc -o saveyourself saveyourself.c -lXm -lXt
                                                          * /
#include <Xm/Protocols.h>
#include <Xm/Label.h>
void saveYourSelfCallback(Widget w, XtPointer client_data,
                             XtPointer call_data)
    printf("Update state file if needed, then update WM_COMMAND\n");
void main(int argc, char** argv)
   Widget toplevel, label;
   XtAppContext app_context;
   Atom WM_SAVE_YOURSELF;
   toplevel = XtAppInitialize(&app_context, "SaveYourSelf",
                               NULL, 0, &argc, argv, NULL, NULL, 0);
   label = XmCreateLabel(toplevel, "saveme", NULL, 0);
   XtManageChild(label);
   WM_SAVE_YOURSELF = XmInternAtom( XtDisplay(toplevel),
                                       "WM_SAVE_YOURSELF",
                                       FALSE);
```

"Handling the Window Manager Save Yourself Protocol" on page 77 describes what your application needs to do to support session management. "Session Management" in Chapter 3 of the *IRIX Interactive User Interface Guidelines* provides further guidelines for handling session management.

Reference

For more information, see *Inter-Client Communication Conventions Manual* (ICCCM). The ICCCM is reprinted as an appendix in O'Reilly and Associates, *X Protocol Reference* Manual, Volume Zero.

Desk Management

Users can use "desks" to create multiple virtual screens. They can assign any primary or support window to any desk, causing that window to appear in the thumbnail sketch in the Desks Overview window.

"Desks" in Chapter 3 of the *IRIX Interactive User Interface Guidelines* discusses the important development concerns issues relating to desks. Review the information in "Session Management," and adhere to the "Session Management Guidelines," and the window manager will take care of desks for you.

The key points to keep in mind are:

- Transient windows appear on every desk and are not shown in the Desks overview window—so choose your transient windows carefully.
- Application windows that are on a desk other than the current one are in a state similar to the minimized state—processing continues although the window is no longer mapped to the screen display. Keep this in mind when selecting which operations should continue to be processed when your application is in a minimized state.
- Users can select different backgrounds for different desks, so your application should not create its own screen background.

Further Reading on Window and Session Management

For more information on window and session management with 4Dwm, refer to the mwm(1X) and 4Dwm(1X) reference pages. You may also want to look at *IRIS Essentials*, since this book explains important window and session management features to your users.

For more information on window and session management with Xt, refer to the chapters on Interclient Communication in these manuals:

- The X Window Systems Programming and Applications with Xt, OSF/Motif Edition, Second Edition, by Doug Young
- O'Reilly Volume Four, X Toolkit Intrinsics Programming Manual, OSF/Motif Edition, by Adrian Nye and Tim O'Reilly

For more information on window and session management with Xlib, refer to the chapters on Inter-Client Communication in O'Reilly Volume One, *Xlib Programming Manual*, by Adrian Nye. For more detailed information, refer to the *Inter-Client Communications Conventions Manual* (ICCCM). (The ICCCM is reprinted as an appendix of O'Reilly Volume Zero, *X Protocol Reference Manual*.)

More detailed information on window properties is available in the *OSF/Motif Programmer's Guide*, in the chapter on "Inter-Client Communication Conventions."

Implementing an Application Model

"Application Models" in Chapter 6 of the *IRIX Interactive User Interface Guidelines* describes four application models based on four different window categories: main primary windows, co-primary windows, support windows, and dialogs. It also describes how to select a model appropriate for your application. This section provides suggestions for implementing each application model, including recommended shell types for your primary windows. "Interacting With the Window and Session Manager" on page 68 describes how to create the windows and get them to look and behave in the manner described in "Application Window Categories and Characteristics" in Chapter 3 of the *IRIX Interactive User Interface Guidelines*.

Implementing the "Single Document, One Primary" Model

This model is the simplest to implement. You can use the ApplicationShell returned by XtAppInitialize() as your application's main window. This model requires no special treatment to handle schemes or for window or session management.

Implementing the "Single Document, Multiple Primaries" Model

The simplest way to implement this model is to use the ApplicationShell returned by XtAppInitialize() as your application's main window. You can create co-primary windows as popup children of the main window using TopLevelShells. This approach requires no special treatment to handle schemes or for window or session management.

You can also choose the implement this model using the techniques described in "Implementing the "Multiple Document, No Visible Main" Model," although this requires more work.

Caution: Don't use XtAppCreateShell() to create co-primary windows. If you do, the windows don't pick up the resources specified in schemes.

Implementing the "Multiple Document, Visible Main" Model

Once again, the simplest way to implement this model is to use the ApplicationShell returned by XtAppInitialize() as your application's main window. You can create co-primary windows as popup children of the main window using TopLevelShells. This approach requires no special treatment to handle schemes or for window or session management.

You can also choose the implement this model using the techniques described in "Implementing the "Multiple Document, No Visible Main" Model," although this requires more work.

Caution: Don't use XtAppCreateShell() to create co-primary windows. If you do, the windows don't pick up the resources specified in schemes.

Implementing the "Multiple Document, No Visible Main" Model

This model requires more careful consideration than the other models. Presumably, the visible windows can be created and destroyed in any order; therefore it is very difficult to use one as a main window and have the others be children of it.

Instead, the best solution in this case is to leave the ApplicationShell returned by XtAppInitialize() unrealized. You can then create the visible co-primary windows as popup children of this invisible shell.

Session management requires a realized ApplicationShell widget so that your application can store restart information in its **XmNargv** and **XmNargc** resources. Because your application's visible windows can be created and destroyed dynamically, you should use ApplicationShells rather than TopLevelShells for your visible windows. Then you can set the **XmNargv** and **XmNargc** resources on any of them. (Another option would be to use TopLevelShells for the visible windows and then explicitly create and set WM_COMMAND and WM_MACHINE properties on the windows.)

One complication when using ApplicationShells is that by default, IRIS IM automatically quits an application when it destroys an ApplicationShell. To avoid this, you must set each window's **XmNdeleteResponse** resource to XmDO_NOTHING, and then explicitly handle the window manager's WM_DELETE_WINDOW protocol for each window. "Handling the Window Manager Delete Window Protocol" on page 76 describes how to implement these handlers.

Another complication is that the initial values of the **XmNargv** and **XmNargc** resources are stored in the application's invisible main window rather than a visible window. This is also true for the **XmNgeometry** resource if specified by the user. To avoid this, you should copy these values from the invisible main window to your application's first visible window.

Caution: Don't use XtAppCreateShell() to create co-primary windows. If you do, the windows don't pick up the resources specified in schemes.

Interacting With the Window and Session Manager

Most communication between an application and a window manager takes place through properties on an application's top-level windows. The window manager can also generate events that are available to the application. You can use Xlib functions to set properties and handle window manager events.

In IRIS IM, shell widgets simplify communications with the window manager. The application can set most window properties by setting shell resources. Shells also select for and handle most events from the window manager.

Because this guide assumes that you are programming in IRIS IM rather than Xlib, this chapter describes the IRIS IM mechanisms for creating windows and interacting with the window and session manager. Topics include:

- "Creating Windows and Setting Decorations"
- "Handling Window Manager Protocols"
- "Setting the Window Title"
- "Controlling Window Placement and Size"

For detailed information about setting window properties using shell resources, consult Chapter 11, "Interclient Communication," in O'Reilly's *X Toolkit Intrinsics Programming Manual* and Chapter 16, "Interclient Communication," in the *OSF/Motif Programmer's Guide*. For detailed information about window properties and setting them using Xlib routines, consult Chapter 12, "Interclient Communication," in O'Reilly's *Xlib Programming Manual*.

Creating Windows and Setting Decorations

Chapter 6, "Application Windows," in the *IRIX Interactive User Interface Guidelines* describes several application models based on four different window categories: main primary windows, co-primary windows, support windows, and dialogs. This section describes how to implement these window categories with proper window decorations and window menu entries:

- "Creating a Main Primary Window"
- "Creating a Co-Primary Window"
- "Creating a Support Window"

• "Creating a Dialog"

To properly integrate with the IRIX Interactive Desktop, you need to use the appropriate shell widget for each widow category. This section describes which shell widget to use for each window category. Then you need to properly set the shell's

XmNmwmFunctions resource to control which entries appear in the window menu and the **XmNmwmDecorations** resource to remove the window's resize handles, if appropriate.

Creating a Main Primary Window

Your application's main primary window must be an ApplicationShell. Typically, you use the ApplicationShell widget returned by XtAppInitialize() as your application's main primary window.

You should set the main primary window's **XmNmwmFunctions** resource to remove the "Close" option from the window menu. Also, if you don't want the user to be able to resize the window, you should set **XmNmwmFunctions** to remove the "Size" and "Maximize" options and set **XmNmwmDecorations** to remove the resize handles. Example 5-2 shows how you can create a main primary window and set the resource values appropriately.

"Main and Co-Primary Windows" in Chapter 6 of the *IRIX Interactive User Interface Guidelines* provides guidelines for using main primary windows.

Example 5-2 Creating a Main Primary Window

```
#include <Xm/Xm.h>
                           /* Required by all Motif applications */
#include <Xm/MwmUtil.h>
                          /* Required to set window menu and decorations */
#include <X11/Shell.h>
                          /* Shell definitions */
void main ( int argc, char **argv )
   Widget mainWindow; /* Main window shell widget */
   XtAppContext app; /* An application context, needed by Xt */
          args[10]; /* Argument list */
                         /* Argument count */
   int
              n;
    * Initialize resource value flags to include all window menu options and
    * all decorations.
    * /
```

}

```
long functions = MWM_FUNC_ALL;
long handleMask = MWM_DECOR_ALL;
n = 0;
 * The following lines REMOVE items from the window manager menu.
functions |= MWM_FUNC_CLOSE; /* Remove "Close" menu option */
/* Include the following two lines only if the window is *not* resizable */
functions |= MWM_FUNC_RESIZE; /* Remove "Size" menu option */
functions |= MWM_FUNC_MAXIMIZE;
                                  /* Remove "Maximize" menu option */
XtSetArg(args[n], XmNmwmFunctions, functions); n++;
/* Include the following two lines only if the window is *not* resizable */
handleMask |= MWM_DECOR_RESIZEH; /* Remove resize handles */
XtSetArg(args[n], XmNmwmDecorations, handleMask); n++
 * Initialize Xt and create shell
* /
mainWindow = XtAppInitialize ( &app, "WindowTest", NULL, 0,
                              &argc, argv, NULL, args, n );
/* ... */
```

Creating a Co-Primary Window

Your application's co-primary windows should be ApplicationShells or TopLevelShells. "Implementing an Application Model" on page 65 describes which to choose depending on your application model. The easiest way to implement these windows are as pop-up children of the shell widget returned by XtAppInitialize() (which is typically your application's main primary window).

If the user can't quit the application from a co-primary window, you should set the window's **XmNmwmFunctions** resource to remove the "Exit" option from the window menu. Also, if you don't want the user to be able to resize the window, you should set **XmNmwmFunctions** to remove the "Size" and "Maximize" options and set **XmNmwmDecorations** to remove the resize handles. Example 5-3 shows how you can create a co-primary window and set the resource values appropriately.

Note: The default action when IRIS IM destroys an ApplicationShell is to quit your application. To avoid this if you are using ApplicationShells for your co-primary windows, you must set each window's **XmNdeleteResponse** resource to XmDO_NOTHING, and then explicitly handle the window manager's WM_DELETE_WINDOW protocol for each window. You might want to follow this approach even if you use TopLevelShells for co-primary windows so that you can simply popdown the window instead of deleting it. This can save time if you might redisplay the window later. "Handling the Window Manager Delete Window Protocol" on page 76 describes how to implement these handlers.

"Main and Co-Primary Windows" in Chapter 6 of the *IRIX Interactive User Interface Guidelines* provides guidelines for using co-primary windows.

Example 5-3 Creating a Co-Primary Window

```
* all decorations.
long functions = MWM_FUNC_ALL;
long handleMask = MWM_DECOR_ALL;
/* ... */
n = 0;
 * The following lines REMOVE items from the window manager menu.
/* Remove the "Exit" window menu option if users can *not* quit from this window */
functions |= MWM_FUNC_QUIT;
/* Include the following two lines only if the window is *not* resizable */
functions |= MWM FUNC RESIZE;
                                   /* Remove "Size" menu option */
functions |= MWM_FUNC_MAXIMIZE; /* Remove "Maximize" menu option */
XtSetArg(args, XmNmwmFunctions, functions); n++;
/* Include the following two lines only if the window is *not* resizable */
handleMask |= MWM_DECOR_RESIZEH; /* Remove resize handles */
XtSetArg(args, XmNmwmDecorations, handleMask); n++;
/* You need the following line only if you use an ApplicationShell for the window */
XtSetArg(args, XmNdeleteResponse, XmDO_NOTHING); n++;
/*
 * Assume that the application has already created a main window and assigned its widget
 * to the variable mainWindow
* /
coPrimary = XtCreatePopupShell( "coPrimary", applicationShellWidgetClass,
                               mainWindow, args, n );
/* ... */
```

Creating a Support Window

Support windows are essentially custom dialogs. The easiest way to create a support window is to use XmCreateBulletinBoardDialog() to create a DialogShell containing a BulletinBoard widget, or use XmCreateFormDialog() to create a DialogShell containing a Form widget. You can then add appropriate controls and displays as children of the BulletinBoard or Form.

Another advantage to using a DialogShell for support windows is that they automatically have the proper window menu options and decorations. If you don't want the user to be able to resize the window—and you implemented the support window as a customized dialog—you should set **XmNnoResize** to "TRUE" to remove the "Size" and "Maximize" options and to remove the resize handles. Example 5-4 shows how you can create a support window and set the resource values appropriately.

"Support Windows" in Chapter 6 of the *IRIX Interactive User Interface Guidelines* provides guidelines for using support windows.

Example 5-4 Creating a Support Window

```
#include <Xm/Xm.h>
                           /* Required by all Motif applications */
#include <Xm/MwmUtil.h>
                          /* Required to set window menu and decorations */
#include <X11/Form.h>
                           /* Form definitions */
Widget
            parentWindow; /* Parent window of support window */
            supportWindow; /* Support window */
Widget
            args[10]; /* Argument list */
Arg
int
                           /* Argument count */
/* ... */
n = 0;
/* Include the following line only if the window is *not* resizable */
XtSetArg(args, XmNnoResize, TRUE); n++
supportWindow = XmCreateFormDialog( parentWindow, "supportWindow", args, n );
/* Create the window interface... */
```

Creating a Dialog

The easiest way to create dialogs is to use the IRIS IM convenience functions such as XmCreateMessageDialog() and XmCreatePromptDialog(). These functions automatically set most of the window characteristics required for the IRIX Interactive Desktop environment.

Dialogs automatically have the proper window menu options and decorations. If you don't want the user to be able to resize the dialog, you should set **XmNnoResize** to "TRUE" to remove the "Size" and "Maximize" options and to remove the resize handles. Example 5-5 shows an example of creating a WarningDialog and setting the resource values appropriately.

Chapter 10, "Dialogs," in the IRIX Interactive User Interface Guidelines provides guidelines for using dialogs.

Example 5-5 Creating a Dialog

```
#include <Xm/Xm.h>
                           /* Required by all Motif applications */
#include <Xm/MwmUtil.h>
                           /* Required to set window menu and decorations */
#include <Xm/MessageB.h> /* Warning dialog definitions */
            parentWindow; /* Parent window of dialog */
Widget
Widget
            dialog;
                          /* Dialog */
            args[10];
                          /* Argument list */
Arq
                          /* Argument count */
int
/* ... */
n = 0;
/* Include the following line only if the window is *not* resizable */
XtSetArg(args, XmNnoResize, TRUE); n++
dialog = XmCreateWarningDialog ( parentWindow, "warningDialog", args, n );
```

Handling Window Manager Protocols

This section describes how to handle window manager **protocols**:

- "Handling the Window Manager Quit Protocol"
- "Handling the Window Manager Delete Window Protocol"
- "Handling the Window Manager Save Yourself Protocol"

Protocols allow the window manager to send messages to your application. The window manager sends these messages only if your application registers callback function to handle the corresponding protocols.

Handling the Window Manager Quit Protocol

When a user selects the "Exit" option from a window menu, the window manager sends a Quit message to your application. You should install a callback routine to handle this event. Example 5-6 demonstrates installing such a callback for the window specified by *mainWindow*.

Example 5-6 Handling the Window Manager Quit Protocol

Note: You must install the quit callback for each window that contains an "Exit" option in its window menu. Often the only such window is your application's main primary window.

The operations performed by the callback function should be the same as those that occur when the user quits from within your application (for example, by selecting an "Exit" option from a File menu). Your application can prompt the user to save any files that are open, to perform any other cleanup, or even to abort the quit.

Handling the Window Manager Delete Window Protocol

When a user selects the "Close" option from a window menu, the window manager sends a Delete Window message to your application. How to handle this message depends on whether the window is a co-primary window, a dialog, or support window. (A main primary window should not have a "Close" option on its window menu.)

To handle the Delete Window message with a co-primary window, you should make sure to set the window's **XmNdeleteResponse** resource to XmDO_NOTHING. Otherwise, IRIS automatically deletes the window and, if the window uses an ApplicationShell, quits the application.

The callback you install can ask for user confirmation and can decide to comply or not comply with the request. If it decides to comply, your application can either pop down or destroy the window. If you think that the user might want to redisplay the window later, popping down the window is usually the better choice because your application doesn't have to re-create it later. Example 5-7 shows an example of installing a callback to handle the Delete Window message.

Example 5-7 Handling the Window Manager Delete Window Protocol in Co-Primary Windows

For support windows and dialogs, you typically want to dismiss the window when the user selects "Close." Therefore, the default value of **XmNdeleteResponse**, XmDESTROY, is appropriate. Additionally, you should perform whatever other actions are appropriate for when that support window or dialog is dismissed. Typically, you can accomplish this by invoking the callback associated with the **Cancel** button, if it exists. Example 5-8 shows an example of this.

Example 5-8 Handling the Window Manager Delete Window Protocol in Support Windows and Dialogs

Handling the Window Manager Save Yourself Protocol

The "Save Yourself" protocol is part of the session management mechanism. The session manager sends a Save Yourself message to allow your application to update the command needed to restart itself in its current state. Currently, the session manager sends Save Yourself messages before ending a session (that is, logging out) and periodically while a session is active.

Your application doesn't need to subscribe to the Save Yourself protocol. Instead, your application can simply update the **XmNargv** and **XmNargc** resources on one of its ApplicationShells whenever it changes state, for example, when it opens or closes a file. The session manager re-saves its state information whenever your application changes these resources. (Actually, the session manager monitors the WM_COMMAND and WM_MACHINE properties, which are set by the ApplicationShell whenever you change its **XmNargv** and **XmNargc** resources.)

If you decide to use Save Yourself for session management, you can handle the protocol on any realized ApplicationShell. Don't use Save Yourself with the unrealized main

window of the "Multiple Document, No Visible Main" application model. When the window manager sends a Save Yourself message to your application, your application must update the value of the **XmNargv** and **XmNargc** resources to specify the command needed to restart the application in its current state. Once you've updated the **XmNargv** and **XmNargc** resources, the session manager assumes that it can safely kill your application. Example 5-9 shows how to handle Save Yourself messages.

Note: Your application shouldn't prompt the user for input when it receives a Save Yourself message.

Example 5-9 Handling the Window Manager "Save Yourself" Protocol

```
/* saveyourself.c
/* Example code for handling the window manager's
                                                         * /
/* WM_SAVE_YOURSELF Protocol
/*
/* cc -o saveyourself saveyourself.c -lXm -lXt
                                                         * /
#include <Xm/Protocols.h>
#include <Xm/Label.h>
void saveYourSelfCallback(Widget w, XtPointer client_data,
                             XtPointer call_data)
   printf("Update WM_COMMAND or state file\n");
void main(int argc, char** argv)
  Widget toplevel, label;
  XtAppContext app_context;
  Atom WM_SAVE_YOURSELF;
   toplevel = XtAppInitialize(&app_context, "SaveYourSelf",
                              NULL, 0, &argc, argv, NULL, NULL, 0);
  label = XmCreateLabel(toplevel, "saveme", NULL, 0);
  XtManageChild(label);
  WM_SAVE_YOURSELF = XmInternAtom( XtDisplay(toplevel),
                                      "WM_SAVE_YOURSELF",
```

```
FALSE);

XmAddWMProtocolCallback( toplevel, WM_SAVE_YOURSELF, saveYourSelfCallback, NULL );

XtRealizeWidget(toplevel);

XtAppMainLoop(app_context);
```

Your application might not be able to fully specify its state using command line options. In that case, you can design your application to create a state file to save its state and to read the state file when it restarts.

Setting the Window Title

To set the title of a main primary window or co-primary window in your application, set the window's **title** resource. If the title you specify uses a non-default encoding, remember to also set the value of the **titleEncoding** resource appropriately. For support windows and dialogs, set the value of the **XmNdialogTitle** resource.

Choose the title according to the guidelines in the section "Window Title Bar" in Chapter 3 of the *IRIX Interactive User Interface Guidelines*. Update the label so that it always reflects the current information. For example, if the label reflects the name of the file the user is working on, you should update the label when the user opens a different file.

Controlling Window Placement and Size

Users have the option of specifying window placement and size, either through the <code>-geometry</code> option interactively using the mouse, or having applications automatically place their windows on the screen. To support automatic window placement, your application should provide default placement information for its main primary and co-primary windows. (Support windows and dialogs appear centered over their parent widget if the value of their <code>XmNdefaultPosition</code> resources are TRUE, which is the default.) You can also specify a default window size, minimum and maximum aspect ratios, and resizing increments for your windows. Typically, you should set these resources in your application's <code>app-default</code> file.

Controlling Window Placement

You should provide initial values for the window shell's x and y resources before mapping the window to specify its default location. The window manager ignores these values if the user requests interactive window placement or specifies a location using the -geometry option when invoking your application. You should not use the window's XmNgeometry resource to control initial window placement, either in your application's source code or its app-default file.

"Window Placement" in Chapter 3 of the *IRIX Interactive User Interface Guidelines* provides guidelines for controlling window placement.

Controlling Window Size

If the user doesn't specify a window size and you don't explicitly set the window size in your application, the initial size of the window is determined by geometry management negotiations of the shell widget's descendents. Typically, the resulting size is just large enough for all of the descendent widget to fit "comfortably." Optionally, you can specify a default initial size for a window by providing initial values for the window's **width** and **height** resources before mapping the window. You should not use the window's **XmNgeometry** resource to control initial window size, either in your application's source code or its app-default file.

You can also set several shell resources to specify minimum and maximum window sizes, minimum and maximum aspect ratios, and resizing increments for a window:

minHeight and minWidth

The desired minimum height and width for the window.

maxHeight and maxWidth

The desired maximum height and width for the window.

minAspectX and minAspectY

The desired minimum aspect ratio (X/Y) for the window.

maxAspectX and maxAspectY

The desired maximum aspect ratio (X/Y) for the window.

baseHeight and baseWidth

The base for a progression of preferred heights and widths for the window. The preferred heights are **baseHeight** plus integral multiples

of **heightInc**, and the preferred widths are **baseWidth** plus integral multiples of **widthInc**. The window can't be resized smaller or larger than the values of the **min*** and **max*** resources.

heightInc and widthInc

The desired increments for resizing the window.

"Window Size" in Chapter 3 of the *IRIX Interactive User Interface Guidelines* provides guidelines for controlling window size.

Customizing Your Application's Minimized Windows

Users can minimize (stow) your application's window on the Desktop, by clicking the minimize button in the top right corner of the window frame or by selecting "Minimize" from the Window Menu. When a window is minimized, it is replaced by a 100 x 100 pixel representation with an identifying label of 13 characters or less. This is referred to as the **minimized** or **stowed** window. (It is also commonly called an icon, but this document uses the term minimized window to prevent confusing it with the Desktop icon.)

This chapter explains how to put the image of your choice on a minimized window. It contains these sections:

- "Some Different Sources for Minimized Window Images" discusses different sources from which you can generate a minimize icon picture.
- "Creating a Minimized Window Image: The Basic Steps" gives a step-by-step explanation of how to customize your minimize icon.
- "Setting the Minimized Window Label" on page 89 describes how to set the label of your minimized window.
- "Changing the Minimized Window Image" on page 89 mentions some special considerations if you want to change the image in your minimized window while your application is running.

007-2006-130

Some Different Sources for Minimized Window Images

You can make a minimized window image out of any image that you can display on your workstation monitor. This means that you can create a picture using showcase or the drawing/painting tool of your choice, or you can scan in a picture, or you can use the capture tool to capture of some portion of your application. You can even have an artist design your minimized window for you. "Choosing an Image for Your Minimized Window" in Chapter 3 of the IRIX Interactive User Interface Guidelines provides some guidelines for designing minimized window images.

Figure 6-1 shows some different minimized window images that were created in different ways. From left to right: the top row shows a scanned-in photograph, a mediarecorder snapshot of the application itself, a scanned-in photograph that was altered with imp, and scanned-in line art; the bottom row shows a drawing representing the application, scanned-in line art, and two artist-designed images.

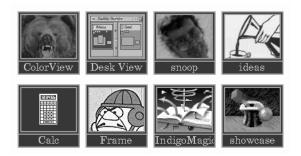


Figure 6-1 Minimized Window Image Examples

Creating a Minimized Window Image: The Basic Steps

It's important for users to be able to easily identify your application's windows when they are minimized, so you should define a specific image and label for each primary and support window in your application. For guidelines on selecting minimize images, see "Choosing an Image for Your Minimized Window" in Chapter 3 of the *IRIX Interactive User Interface Guidelines*.

To make a minimized window image for your application:

- 1. Create an RGB image. If your image is already in RGB format, then all you have to do is resize the image to an appropriate size (look at the setting of the iconImageMaximum resource in 4Dwm to see the maximum size of the stow icon, currently 85x67). See "Resizing the RGB Image Using imgworks" on page 88 for instructions on resizing the image.
 - If your image is not in RGB format, you must convert it to RGB. One way to do this is to use the Media Recorder tool to obtain your image. See "Using mediarecorder to Get an RGB Format Image" on page 86 for instructions.
- 2. Scale the image to the correct size. See "Resizing the RGB Image Using imgworks" on page 88 for instructions.
- 3. Name the image file. The filename should consist of two parts:
 - The application class name (technically, the res_name field of the WM_CLASS property). You can determine the class name by using xprop on your application window and looking at the WM_CLASS property.
 - The .icon suffix.

This gives you a name of the form res_name.icon. For example, if your application's name is "chocolate," the name of your image file should be:

chocolate.icon

4. Put the file in the /usr/lib/images directory.

Using mediarecorder to Get an RGB Format Image

You can use the mediarecorder tool to capture an image on the screen. To invoke mediarecorder, enter:

% mediarecorder

An example of the mediarecorder tool is shown in Figure 6-2.

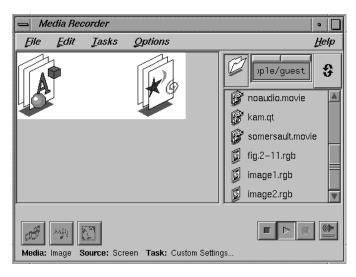


Figure 6-2 The mediarecorder Tool

Taking a Screen Snapshot With mediarecorder

You can create a new image file of your minimized window by taking a snapshot of it on the screen.

- 1. Display the desired image on the screen.
- 2. Click the Image menu button and choose "Still Image from Screen."

- 3. Choose one of the following from the "Still Image from Screen" rollover menu:
 - Select Area...: Allows you to hold down the left mouse button and drag out a selection outline to mark the area you wish to record.
 - Window Area...: Allows you to select a particular window on the screen to record. When you choose this menu item, the cursor changes to resemble a camera. To choose a window to record, simply click inside of it.

A dashed outline appears on the screen: the space inside the outline is what will be captured.

Note: To **cancel** any selection, position the cursor over an edge of the selection outline and click the right mouse button. To bring the selection outline back once it is cancelled, select Edit menu and choose "Show Selection Frame."

4. Open the Tasks menu, choose Image, and select Screen Snapshot from the Image rollover menu. The snapshot adjusts the capturing file format and frame size so that they are ideally suited for a screen snapshot.

Tip: Once you choose Screen Snapshot, you can view those specific settings by selecting "Show Task Settings" from the Tasks menu.

5. After you select the area, click the Record button.

If you have the Clip Bin open, the image filename appears in the clip bin as soon as the snapshot has been taken.

Your file is given a default name and saved in the directory from which you launched Media Recorder.

6. To see the image, click the Play button.

If the image looks good, then you're ready to resize it. See "Resizing the RGB Image Using imgworks" on page 88 for instructions.

See the mediarecorder(1) reference page for more information about using mediarecorder. The mediarecorder is described fully in the *Digital Media Tools Guide*.

Resizing the RGB Image Using imgworks

You can use imgworks to resize your RGB image to the appropriate size for a minimized window image. The maximum size is determined by the value of the **iconImageMaximum** resource in 4Dwm, which is currently 85x67.

To find the imgworks icon, select "File QuickFind" from the Find toolchest. When the QuickFind window appears, type:

imgworks

into the text field. The imgworks icon appears in the drop pocket. Drag the icon to the Desktop and drop it. Then run imgworks by double-clicking the icon.

To resize your image using imgworks, follow these steps:

- 1. Open your image file by selecting "Open" from the File menu and selecting your file from the Image Works: Open Image... window. Your image appears in the main window.
- 2. To scale the image, select "Scale..." from the Transformations menu. The Image Works: Scale window appears.
- 3. Scale the image by typing in an appropriate scale factor. The dimensions of the new image (in pixels) are listed in the Scale window.
- 4. When you're happy with the dimensions listed in the Scale window, click the **Apply** button. The resized image appears in the main window. Save it by selecting "Save" from the File menu.

Refer to the imgworks(1) reference page for more information on imgworks.

Setting the Minimized Window Label

By default, the **4Dwm** window manager reuses the title bar label for the minimized window label. To explicitly set the label of the minimized window, change the value of the window's **XmNiconName** resource. For example, in your application's /usr/lib/X11/app-defaults file, type:

appname*iconName: myApp

"Labeling a Minimized Window" in Chapter 3 in the *IRIX Interactive Desktop User Interface Guidelines* lists guidelines for choosing a label.

Changing the Minimized Window Image

Your application can also change its minimized window's image while it is running (for example, to indicate application status) by setting the window's **XmNiconWindow** resource. However, it can be very difficult to handle color images without causing visual and colormap conflicts. If you decide to change the image, the image you install should:

- Use the default visual.
- 2. Use the existing colormap without creating any new colors (preferably, your image should use only the first 16 colors in the colormap). This potentially implies dithering or color quantization of your image.

Note: The 4Dwm window manager automatically handles your application's initial minimized window image (that is, the image automatically loaded from the /usr/lib/images directory at application start-up). If you don't want to change this image while your application is running, your application doesn't need to do anything to support displaying the image properly.

007-2006-130

Interapplication Data Exchange

This chapter describes how to implement the recommended data exchange mechanisms in your applications. It contains these sections:

- "Data Exchange Overview" on page 91 provides a brief description of how the Primary and Clipboard Transfer Models should work in your application. You should implement both.
- "Implementing the Primary Transfer Model" on page 95 describes how to implement the Primary Transfer Model in your application.
- "Implementing the Clipboard Transfer Model" on page 98 describes how to implement the Primary Transfer Model in your application.
- "Supported Target Formats" on page 101 provides tables listing the atom names of supported data formats, along with brief descriptions of what each format is used for.
- "Data Conversion Service" on page 105 describes the service available for converting files from one data format to another. For details on the process, refer to Appendix G, "Using GoldenGate Data Conversion Services".

Data Exchange Overview

As detailed in Chapter 5, "Data Exchange on the IRIX Interactive Desktop," in the *IRIX Interactive User Interface Guidelines*, Silicon Graphics recommends that your application support both the Primary and Clipboard Transfer Models. The Primary Transfer Model allows users to copy data using mouse buttons, whereas the Clipboard Transfer model allows users to use the "Cut," "Copy," and "Paste" options from the Edit menu (or the corresponding keyboard accelerators) to transfer data.

Note: Silicon Graphics recommends that you not use the IRIS IM clipboard routines for handling data exchange.

The data exchange model recommended by Silicon Graphics is based on the standard mechanisms provided by the X and Xt. You can consult the O'Reilly & Associates book *The X Window System, Volume 4: X Toolkit Intrinsics Programming Manual* by Adrian Nye for more information on the standard Xt data exchange methods.

The following sections describe:

- "Primary Transfer Model Overview"
- "Clipboard Transfer Model Overview"
- "Interaction Between the Primary and Clipboard Transfer Models"

Primary Transfer Model Overview

When the user selects some data in an application, the application should highlight that data and assert ownership of the PRIMARY selection. Until the application loses the PRIMARY selection, it should then be prepared to respond to requests for the selected data in various target formats. "Supported Target Formats" on page 101 describes the standard target formats.

When the user selects data in another application, your application loses ownership of the PRIMARY selection. 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** discussed in Section 4.2, "Selection Actions," in the *OSF/Motif Style Guide*, *Release 1.2*. There is an exception to this guideline: those applications that use selection only for primary transfer, for example, the **winterm** 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, "Selection Actions," in the *OSF/Motif Style Guide*, *Release 1.2*.

The **persistent always selection** mechanism allows the user to have data selected in different applications. The user can still manipulate selected data using application controls. Furthermore, the user can reassert the selected data as the PRIMARY selection by pressing **<Alt-Insert>**.

When the user clicks the middle mouse button (**BTransfer**) in your application, your application should attempt to copy the primary selection to the current location of the mouse pointer. First, your application should request a list of target formats supported by the primary selection owner. Then your application should select the most appropriate target format and request the primary selection in that format.

"Supporting the Primary Transfer Model" in Chapter 5 of the *IRIX Interactive User Interface Guidelines* further discusses use of the Primary Transfer Model.

Clipboard Transfer Model Overview

When the user selects the "Copy" option from your application's Edit menu (or uses the keyboard accelerator), your application should assert ownership of the CLIPBOARD selection. Until the application loses the CLIPBOARD selection, it should then be prepared to respond to requests for the data selected at the time your application took ownership of the CLIPBOARD selection. (In other words, your application must somehow store the value of the selection when the user performs the copy action; the application can then provide this value even if the user subsequently changes the application's selection.)

When the user selects the "Cut" option for your application's Edit menu (or uses the keyboard accelerator), your application should assert ownership of the CLIPBOARD selection. Your application must cut the selected data, but it should store the data and be prepared to respond to requests for the data until it loses ownership of the CLIPBOARD selection.

When the user selects the "Paste" option for your application's Edit menu (or uses the keyboard accelerator), your application should attempt to copy the clipboard selection to the current location of the location cursor. First, your application should request a list of target formats supported by the clipboard selection owner. Then your application should select the most appropriate target format and request the clipboard selection in that format.

"Supporting the Clipboard Transfer Model" in Chapter 5 of the *IRIX Interactive User Interface Guidelines* further discusses use of the Clipboard Transfer Model.

Interaction Between the Primary and Clipboard Transfer Models

Silicon Graphics recommends that you implement the Primary and Clipboard Transfer Models so that they operate separately. The only complication is maintaining data in the PRIMARY selection when the user performs a cut action. Consider the following example:

- 1. The user selects data in an application. The application asserts ownership of the PRIMARY selection.
- 2. The user performs a cut action. The application asserts ownership of the CLIPBOARD selection and removes the selected data from the display.
- 3. The user goes to another application that already has data selected.
- 4. The user cuts the data selected in the second application. The second application asserts ownership of the CLIPBOARD selection and removes the selected data from the display.

The clipboard actions described above should not affect the PRIMARY selection. In this example, the first application should retain ownership of the PRIMARY selection and continue to be prepared to respond to requests for the value of the PRIMARY selection. To support this, the application should somehow store the value of the PRIMARY selection until it no longer owns the PRIMARY selection.

To properly handle the situation described above, your application should implement the following:

- 1. In the function that handles the Clipboard Transfer Model's cut action, test to see whether the application owns the PRIMARY selection. If it does, you should preserve the selected data. If selections in your application are typically small (for example, ASCII text), you might simply copy the data to a buffer. If selections in your application are typically large (for example, sound or movie clips), you might remove the data from the display but retain pointers to it.
- In the function that handles losing the PRIMARY selection, test to see whether you have data preserved from a cut action. If so, and the application currently doesn't own the CLIPBOARD selection, you should free that data or reset the pointers to it.

Implementing the Primary Transfer Model

This section describes how to implement support for the Primary Transfer Model in your application. Topics covered include:

- "Data Selection"
- "Requests for the Primary Selection"
- "Loss of the Primary Selection"
- "Inserting the Primary Selection"

Note: Silicon Graphics recommends that you don't use the IRIS IM clipboard routines, because they are not as flexible as the Xt selection routines.

Data Selection

When the user selects data in a window of your application, it should call XtOwnSelection(3Xt) to assert ownership of the PRIMARY selection and highlight the selected data.

The code fragment in Example 7-1 shows a simple example of asserting ownership of the PRIMARY selection. For clarity, this example omits code for manipulating the selection itself (for example, setting up pointers to the selection).

"Selection" in Chapter 7 of the *IRIX Interactive User Interface Guidelines* discusses guidelines for allowing users to select data and for hightlighting selected data.

Example 7-1 Asserting Ownership of PRIMARY Selection

```
Boolean ownPrimary;

/*
    w is window in which selection occurred
    event is pointer to event that caused selection
*/

void dataSelected(Widget w, XButtonEvent *event)
{
...
```

```
Assert ownership of PRIMARY selection.
  XA_PRIMARY is the slection.
   event->time is timestamp of the event.
  primaryRequestCallback is the function called
       whenever another application requests the
       value of the PRIMARY selection.
   lostPrimaryCallback is the function called whenever
       the application loses the selection.
* /
ownPrimary = XtOwnSelection(w, XA_PRIMARY, event->time,
                            primaryRequestCallback,
                            lostPrimaryCallback,
                            NULL);
   If we successfully obtained ownership, highlight
   the data; otherwise, clean up
if (ownPrimary)
 highlightSelection();
else
  lostPrimaryCallback(w, XA_PRIMARY);
```

Requests for the Primary Selection

When you assert ownership of the PRIMARY selection, one of the parameters you pass to XtOwnSelection() is a callback function to handle requests for the value of the PRIMARY selection. When another application requests the value of the PRIMARY selection, the Xt selection mechanism invokes your application's callback function.

The requesting application indicates a desired target format. Typically, a requestor first asks for the special target format TARGETS. Your application should respond with a list of target formats it supports. The requestor then chooses an appropriate target format and requests the selection value in that format. "Supported Target Formats" on page 101 describes some of the common target formats your application should support.

Loss of the Primary Selection

When your application loses the PRIMARY selection and your application follows the **persistent always selection** model discussed in "Primary Transfer Model Overview" on page 92, don't remove the highlight from any selected data. The user should still be able to cut or copy any selected data using the Clipboard Transfer Model. If your application follows the **nonpersistent selection** model as discussed in "Primary Transfer Model Overview," you should remove the highlight.

Your application should also test to see whether you have data preserved from a cut action (see "Cut Actions" on page 98). If so, and your application currently doesn't own the CLIPBOARD selection, you should free that data or reset the pointers to it. "Interaction Between the Primary and Clipboard Transfer Models" on page 94 describes the rationale for this procedure.

Note: To comply with the *IRIX Interactive User Interface Guidelines*, if the user presses **<Alt-Insert>** in your application, you should reassert ownership of PRIMARY for your application.

Inserting the Primary Selection

When the user clicks the middle mouse button in your application, it should perform the steps described below.

- Your application should ask the owner of the PRIMARY selection for a list of its TARGETS, using XtGetSelectionValue() with selection PRIMARY and target TARGETS.
- 2. Your application should look through the list of supported targets, select the one that is appropriate for your application, and call XtGetSelectionValue() again with that new target.
- 3. If the selection owner does not support TARGETS, then your application should ask for the target STRING, if it can support that target.
 - Silicon Graphics recommends that you support STRING, even if your application doesn't support text. For instance, a movie player could get the selection as a string and try to parse it as a filename. That way users could select a filename in a terminal emulator window and paste it into another application.

Implementing the Clipboard Transfer Model

This section describes how to implement support for the Clipboard Transfer Model in your application. Topics include:

- "Cut Actions"
- "Copy Actions"
- "Requests for the Clipboard Selection"
- "Paste Actions"
- "Loss of the Clipboard Selection"

Cut Actions

When the user performs a cut action, your application should:

- 1. Call XtOwnSelection(3Xt) to assert ownership of the CLIPBOARD selection.
- 2. Remove the selected data from the display. Retain the selected data until your application loses ownership of the CLIPBOARD selection.
- 3. Test to see whether the application owns the PRIMARY selection. If it does, you should preserve the selected data, even after losing ownership of the CLIPBOARD selection. You should retain the data until your application also loses ownership of the PRIMARY selection.

If selections in your application are typically small (for example, ASCII text), you might simply copy the data to a buffer. If selections in your application are typically large (for example, sound or movie clips), you might remove the data from the display but retain pointers to it.

The code fragment in Example 7-2 shows a simple example of handling a cut action and asserting ownership of the CLIPBOARD selection. For clarity, this example omits code for manipulating the selection itself (for example, setting up pointers to the selection).

Example 7-2 Handling Cut Actions in the Clipboard Transfer Model

```
Boolean ownPrimary;
Boolean primaryPreserved;
/*
   w is window in which selection occurred
   event is pointer to event that caused selection
```

```
* /
void selectionCut(Widget w, XButtonEvent *event)
  /*
     Assert ownership of CLIPBOARD selection.
     XA_CLIPBOARD is the selection.
     event->time is timestamp of the event.
     clipboardRequestCallback is the function called
         whenever another application requests the
         value of the CLIPBOARD selection.
     lostClipboardCallback is the function called whenever
         the application loses the selection.
  * /
  ownClipboard = XtOwnSelection(w, XA_CLIPBOARD, event->time,
                                clipboardRequestCallback,
                                lostClipboardCallback,
                                NULL);
  if (ownClipboard)
       Retain the selected data until the application loses
       ownership of the CLIPBOARD selection.
    * /
    preserveClipboardSelection();
    /*
       If we also own the PRIMARY selection, we need to
       preserve the selected data separately so that we can
       continue to satisfy requests for the PRIMARY selection
       even if we lose the CLIPBOARD selection.
    * /
    if (ownPrimary)
      primaryPreserved = preservePrimarySelection();
. . .
```

Copy Actions

When the user performs a copy action, your application should call XtOwnSelection(3Xt) to assert ownership of the CLIPBOARD selection. No other actions are required.

Requests for the Clipboard Selection

When you assert ownership of the CLIPBOARD selection, one of the parameters you pass to XtOwnSelection() is a callback function to handle requests for the value of the CLIPBOARD selection. When another application requests the value of the CLIPBOARD selection, the Xt selection mechanism invokes your application's callback function.

The requesting application indicates a desired target format. Typically, a requestor first asks for the special target format TARGETS. Your application should respond with a list of target formats it supports. The requestor then chooses an appropriate target format and requests the selection value in that format. "Supported Target Formats" on page 101 describes some of the common target formats your application should support.

Paste Actions

When the user selects "Paste" from the File menu, your application should:

- 1. Ask the owner of the CLIPBOARD selection for a list of its TARGETS, using XtGetSelectionValue() with selection CLIPBOARD and target TARGETS.
- 2. Look through the list of supported targets, select the one that is appropriate for your application, and call XtGetSelectionValue() again with that new target.
- 3. If the selection owner doesn't support TARGETS, then your application should ask for the target STRING, if it can support that target.

Silicon Graphics recommends that you support STRING, even if your application doesn't support text. For instance, a movie player can get the selection as a string and try to parse it as a filename. That way users can select a filename in a terminal emulator window and paste it into another application.

Loss of the Clipboard Selection

When your application loses the Clipboard selection, don't remove the highlight from any selected data. The user should still be able to cut or copy any selected data. Your application can discard any data it had retained as a result of a cut operation (see "Cut Actions" on page 98).

Supported Target Formats

Every application should support the TARGETS, TIMESTAMP, MULTIPLE, and STRING targets. The Xt selection functions support the MULTIPLE targets for you.

XmuConvertStandardSelection() supports the TIMESTAMP target. (Silicon Graphics recommends that applications use XmuConvertStandardSelection() because it also supports HOSTNAME, NAME, CLIENT_WINDOW, and a variety of other useful targets.) Your application must support the TARGETS and STRING targets itself. In addition, Silicon Graphics has defined other targets for data types used by Silicon Graphics applications and libraries.

The tables that follow list supported target formats:

- Table 7-1 lists target names for audio formats
- Table 7-2 shows target names for image formats
- Table 7-3 presents target names for movie formats
- Table 7-4 lists target names for 3D graphics formats
- Table 7-5 shows target names for Silicon Graphics data types
- Table 7-6 shows target names for World Wide Web formats

Table 7-1 Audio Formats

Name of Atom/Target	Description
AIFF_FILE	Audio Interchange Format, used on Apple systems.
AIFF_C_FILE	Modified version of Apple's AIFF, compatible with SGI systems.
NEXT_FILE	Used on Next and Sun systems.
SD2_FILE	Sound Designer 2 format
WAVE_FILE	Microsoft Wave format
MPEG_1_AUDIO_FILE	MPEG Audio. The name of a file that contains MPEG-1 audio data.
MPEG_1_AUDIO	Stream of audio data, in MPEG-1 Stream format.
SGI_AUDIO_FILE	The name of a file that contains Silicon Graphics format sound data, that can be read using libaudiofile. The file is the responsibility of the receiver, once the selection owner has generated it.
SGI_AUDIO	Stream of audio data, readable with the SGI audio library.

Table 7-2Image Formats

Name of Atom/Target	Description
FITS_FILE	Flexible Image Transport System
GIF_89	Graphics Interchange Format (streaming bit format)
GIF_89_FILE	Graphics Interchange Format (file format)
JFIF_FILE	JPEG File Interchange Format
DIB_FILE	Microsoft image format
PHOTO_CD_FILE	Kodak photo CD
PPM_FILE	Portable pixmap format
PNM_FILE	Portable anymap format
PGM_FILE	Portable graymap format

Table 7-2 (continued)	Image Formats
Name of Atom/Target	Description
SGI_RGBIMAGE_FILE	The name of a file that contains a Silicon Graphics format image file. This is an rgb file. The file is the responsibility of the receiver, once the selection owner has generated it.
SGI_RGBIMAGE	The rgb image data stream.

Tagged Image File Format

Table 7-3 Movie Formats

TIFF_FILE

Name of Atom/Torget	Description
Name of Atom/Target	Description
APPLE_QUICKTIME_FILE	Apple Quicktime™ format
AVID_OMFI_FILE	AVID OMFI™
AVI_FILE	Microsoft AVI™ format
MPEG_1_VIDEO_FILE	Motion Picture Experts Group MPEG-1 file
MPEG_1_VIDEO	Stream format
MPEG_1_SYSTEMS_FILE	Motion Picture Experts Group MPEG-1 systems file
MPEG_1_SYSTEMS	Stream format
SGI_MOVIE_FILE	SGI movie format
SGI_MOVIE	Stream format

Table 7-43D Graphics Formats

Name of Atom/Target	Description
INVENTOR_2_1	SGI Open Inventor V2.1 data
INVENTOR_2_1_FILE	SGI Open Inventor V2.1 file
AUTODESK_DXF_FILE	AUTODESK DXF™
AUTODESK_3DS_FILE	AUTODESK 3DS™
ALIAS_FILE	Alias wire file
IGES_FILE	IGES file

Table 7-4 (continued) 3D Graphics Formats

Name of Atom/Target	Description
PIXAR_RIB_FILE	Pixar Renderman .RIB file
SOFTIMAGE_HRC_FILE	Softimage .hrc file
SOFTIMAGE_DSC_FILE	Softimage .dsc file
WAVEFRONT_OBJ_FILE	Wavefront .OBJ file
VRML_1_0_FILE	VRML 1.0 file

Table 7-5 Additional Data Types Supported by Silicon Graphics

Name of Atom/Target	Description
INVENTOR	Synonym for INVENTOR_2_0.
_SGI_RGB_IMAGE_FILENAME	Replaced by SGI_RGBIMAGE_FILE.
_SGI_RGB_IMAGE	Replaced by SGI_RGBIMAGE.
_SGI_AUDIO_FILENAME	Replaced by SGI_AUDIO_FILE.
_SGI_AUDIO	Replaced by SGI_AUDIO.
_SGI_MOVIE_FILENAME	Replaced by SGI_MOVIE_FILE.
_SGI_MOVIE	Replaced by SGI_MOVIE.
SGI_SHOWCASE_FILE	Name of file containing SGI Showcase data.

Caution: Xt implements a timeout when transferring data using the selection mechanism. The default is five seconds. Often, this is inadequate for applications transferring audio, image, or movie data. Therefore, if your application supports receiving such selections, you should call XtAppSetSelectionTimeout() to change the timeout to a larger value.

Table 7-6 World Wide Web Targets

Type of Atom/Target	Description	
HTML	Hypertext Markup Language as an ASCII Stream.	
HTML_FILE	Name of the file containing HTML.	

Note: Silicon Graphics applications should also support the generic X11/ICCCM targets such as STRING and COMPOUND_TEXT.

Data Conversion Service

Silicon Graphics provides GoldenGate data conversion service to help you convert data from one format to another, offloading the responsibility for data conversion from your application. See Appendix G, "Using GoldenGate Data Conversion Services", for more information.

The GoldenGate conversion service consists of four elements:

- the header file, /usr/include/convert/SqCvt.h
- the conversion library, libcvt.so
- the converter registry file, /etc/ConverterRegistry
- a set of pluggable converters

The header file and library provide the Application Programmatic Interface (API) for the service. The converter registry file describes the converters available to the service. The actual code for the converters is stored in separate Dynamic Shared Objects (DSOs). When an application program tries to access a converter, the service dynamically loads the associated DSO.

The GoldenGate API uses Digital Media parameter-value lists to describe data formats. See the *IRIS Media Libraries Programming Guide* for information on how to use the Digital Media library to create and manipulate digital media parameter-value lists.

The Converter Registry

The converter registry contains entries describing each converter available to the service. Each converter entry lists the data format that the converter takes as input, the format that the converter produces as output, and information that allows the service to locate the converter code.

The converter registry can also contain entries defining **pipelines**. A pipeline is a set of converters connected in series, with the output of one converter feeding the next. As far as application programs are concerned, a pipeline is just another converter.

The GoldenGate API

All programs using the GoldenGate API should include the libSgCvt header file: #include <convert/SgCvt.h>

The GoldenGate API uses a fairly simple programming model. For application programs, the most complicated part of the conversion process is picking an appropriate converter.

Once the application program has picked a converter, it initializes the conversion pipeline, and proceeds to send data through the pipeline. The pipeline is a "black box." It may consist of one converter, or several converters running in series. The API for application programs using the service is described in Appendix G, "Using GoldenGate Data Conversion Services".

Converters, on the other hand, must read data from the pipeline, convert it, and write data back to the pipeline. Again, the pipeline is a black box. A converter can't tell if its input is coming from another converter, or from an application program. Likewise, it can't tell if its output is going to another converter or back to the application program. Designing converters is discussed in Appendix G, "Using GoldenGate Data Conversion Services".

Monitoring Changes to Files and Directories

The File Alteration Monitor (FAM) monitors changes to files and directories in the filesystem and notifies interested applications of these changes. Your application can use FAM to get an up-to-date view of the filesystem rather than having to poll the filesystem. This chapter describes the required libraries and provides a basic list of steps for using FAM. For more detailed information, refer to the fam(1M) and FAM(3X) reference pages.

This chapter contains these sections:

- "FAM Overview" on page 107 provides an overview to FAM including the libraries and header files needed to use FAM in your application.
- "The FAM Interface" on page 109 describes the FAM API.
- "Using FAM" on page 118 provides a simple example demonstrating FAM.

FAM Overview

Typically, if applications need to monitor the status of a file or directory, they must periodically poll the filesystem. FAM provides a more efficient and convenient method.

FAM consists of the FAM daemon, fam, and a library for interacting with this daemon. An application can request fam to monitor any files or directories in the filesystem. When fam detects changes to these files, it notifies the application.

This section provides an overview of FAM and describes:

- "Theory of Operation" on page 108
- "FAM Libraries and Include Files" on page 108

Theory of Operation

FAM uses imon, a pseudo device, to monitor filesystem activity on your system on a file-by-file basis. You can refer to the imon(7) reference page for more information on its operation, but you should not attempt to access imon directly.

When you provide FAM with the name of a file or directory to monitor, FAM passes the request to imon, which begins monitoring the inode corresponding to the pathname. When imon detects a change to an inode that it is monitoring, it notifies FAM, which matches the inode to a corresponding filename. FAM then generates a FAM event on a socket. Your application can either monitor the socket or periodically poll FAM to detect FAM events.

This difference between FAM and imon can produce some unexpected results. For example, if a user moves a file, FAM reports that the file is deleted. The reason is that FAM monitors files by name and not inode, so it doesn't know that the file still exists.

Note: Unlike local files and directories, FAM monitors NFS-mounted files and directories by name rather than by inode.

As another example, consider the case where FAM is monitoring a file. If the user deletes the file, FAM correctly reports that fact. Then FAM polls the directory every few seconds to see if the file has been created. If you need to detect the creation of a given file by name, you may want to monitor the directory in which it will be created and watch for FAM events notifying the creation of a file by that name in the directory.

Whenever FAM is asked to monitor a file/directory that resides on a remote (NFS) filesystem, FAM tries to make a connection to the FAM on the NFS server. If it succeeds, it asks the server fam to monitor the file. The server FAM sends FAM events, and the original FAM translates those events to a form its client can use. If FAM can't connect to FAM on the server, it monitors the file itself by polling every few seconds. Polling over NFS has a high overhead.

FAM Libraries and Include Files

The FAM interface routines are in the libfam library. libfam depends on the libC library. Be sure to specify -lfam before -lC in the compilation or linking command. If

you are using fam from a C++ program, libC is included automatically. You must include libC if you are using fam from a C program.

You must include <fam.h> in any source file that uses FAM. You must also include <sys/select.h> if you use the select(2) system call.

The FAM Interface

This section describes the functions you use to access FAM from your application:

- "Opening and Closing a FAM Connection" on page 109
- "Monitoring a File or Directory" on page 110
- "Suspending, Resuming, and Canceling Monitoring" on page 111
- "Detecting Changes to Files and Directories" on page 112
- "FAM Examples" on page 116

Opening and Closing a FAM Connection

The function FAMOpen() opens a connection to fam:

```
int FAMOpen(FAMConnection* fc)
```

FAMOpen() returns 0 if successful and -1 if unsuccessful. FAMOpen() initializes the FAMConnection structure passed to it, which you must use in all subsequent FAM procedure calls in your application.

An element of the FAMConnection structure is the file descriptor associated with the socket that FAM uses to communicate with your application. You need this file descriptor to perform select() operations on the socket. You can obtain the file descriptor using the FAMCONNECTION GETFD() macro:

```
FAMCONNECTION_GETFD (fc)
```

The function FAMOpen2 tells FAM the application's name:

int FAMOpen2(FAMConnection* fc, const char* appName)

FAM uses **appName** when it prints debugging messages.

```
The function FAMClose() closes a connection to fam:
```

```
int FAMClose(FAMConnection* fc)
```

FAMClose() returns 0 if successful and -1 if unsuccessful.

Monitoring a File or Directory

 ${\tt FAMMonitorDirectory()} \ and \ {\tt FAMMonitorFile()} \ tell \ {\tt FAM} \ to \ start \ monitoring \ a \ directory \ or \ file \ respectively:$

FAMMonitorDirectory() monitors not only changes that happens to the contents of the specified directory file, but also to the files in the directory. If the directory contains subdirectories, FAMMonitorDirectory() monitors changes to the subdirectory files, but not the contents of those subdirectories. FAMMonitorFile() monitors only what happens to the specified file. Both functions return 0 if successful and -1 otherwise.

The first argument to these functions is the FAMConnection structure initialized by FAMOpen(). The second argument is the full pathname of the directory or file to monitor. Note that you can't use relative pathnames.

The third argument is a FAMRequest structure that these functions initialize. You can pass this structure to FAMSuspendMonitor(), FAMResumeMonitor(), or FAMCancelMonitor() to respectively suspend, resume, or cancel the monitoring of the file or directory. "Suspending, Resuming, and Canceling Monitoring" on page 111 further describes these functions.

The fourth argument is a pointer to any arbitrary user data that you want included in the FAMEvent structure returned by FAMNextEvent() when this file or directory changes.

FAM then generates **FAM events** whenever it detects changes in monitored files or directories. "Detecting Changes to Files and Directories" on page 112 describes how to detect and interpret these events.

Two similar routines are FAMMonitorDirectory2() and FAMMonitorFile2():

In these routines, the caller picks the request number, not libfam. The caller specifies the request number by putting it in the FAMRequest before calling the routine. For example:

```
FAMConnection fc;
FAMRequest frq;
...
frq.reqnum = some_number_associated_with_tmp;
if (FAMMonitorDirectory2(&fc, "/tmp", &frq) < 0)
    perror("can't monitor /tmp");</pre>
```

If you use the -2 routines, you must choose unique request numbers. See FAMAcknowledge below.

It's up to you to determine which routines to use: the -2 routines or the original routines.

Suspending, Resuming, and Canceling Monitoring

Once you've begun monitoring a file or directory, you can cancel monitoring or temporarily suspend and later resume monitoring.

FAMSuspendMonitor() temporarily suspends monitoring a file or directory. FAMResumeMonitor() resumes monitoring the file or directory. Suspending file monitoring can be useful when your application does not need to display information about a file (for example, when your application is iconified).

Note: FAM queues any changes that occur to the file or directory while monitoring is suspended. When your application resumes monitoring, FAM notifies it of any changes.

The syntax for these functions is:

```
int FAMSuspendMonitor(FAMConnection *fc, FAMRequest *fr);
int FAMResumeMonitor(FAMConnection *fc, FAMRequest *fr);
```

fc is the FAMConnection returned by FAMOpen(), and fr is the FAMRequest returned by either FAMMonitorFile() or FAMMonitorDirectory(). Both functions return 0 if successful and -1 otherwise.

When your application is finished monitoring a file or directory, it should call FAMCancelMonitor():

```
int FAMCancelMonitor(FAMConnection *fc, FAMRequest *fr)
```

FAMCancelMonitor() instructs FAM to no longer monitor the file or directory specified by *fr*. It returns 0 if successful and -1 otherwise.

After you call FAMCancelMonitor(), FAM sends a FAMAcknowledge event. When you've seen the FAMAcknowledge event, you know it's safe to re-use the request number (if you're using the -2 form monitoring routines).

Detecting Changes to Files and Directories

Whenever FAM detects changes in files or directories that it is monitoring, it generates a **FAM event**. Your application can receive FAM events in one of two ways:

The Select approach

Your application performs a select(2) on the file descriptor in the FAMConnection structure returned by FAMOpen(). When this file descriptor becomes active, the application calls FAMNextEvent() to retrieve the pending FAM event.

The Polling approach

Your application periodically calls FAMPending() (typically when the system is waiting for input). When FAMPending() returns with a positive return value, your application calls FAMNextEvent() to retrieve the pending FAM events.

FAMPending() has the following syntax:

```
int FAMPending(FAMConnection *fc)
```

It returns 1 if there is a FAM event queued, 0 if there is no queued event, and -1 if there is an error. FAMPending() returns immediately (that is, it does not wait for an event).

Once you have determined that there is a FAM event queued, whether by using the select or polling approach, call FAMNextEvent() to retrieve it:

```
int FAMNextEvent(FAMConnection *fc, FAMEvent *fe)
```

FAMNextEvent() returns 0 if successful and -1 if there is an error. The first argument to FAMNextEvent() is the FAMConnection structure initialized by FAMOpen(). The second argument is a pointer to a FAMEvent structure, which FAMNextEvent() fills in with information about the FAM event. The format of the FAMEvent structure is:

```
typedef struct {
   FAMConnection* fc;
   FAMRequest fr;
   char *hostname;
   char filename[PATH_MAX];
   void *userdata;
   FAMCodes code;
   } FAMEvent;
```

fc is the FAMConnection structure initialized by FAMOpen().

fr is the FAMRequest structure returned by either FAMMonitorFile() or
FAMMonitorDirectory() when you requested that FAM monitor the file or directory
that changed.

hostname is an obsolete field. Don't use it in your applications.

filename is either the full pathname of the file or directory that you monitored or the name of a file in a directory that you monitored.

userdata is the arbitrary data pointer that you provided when you called either
FAMMonitorFile() or FAMMonitorDirectory() to monitor this file or directory. If
you used the -2 routine, FAMMonitorDirectory2() or FAMMonitorFile2(),
userdata is undefined.

code is an enumerated value of type FAMCodes that describes the change that occurred. It can take any of the following values:

FAMChanged Some value of the file or directory that can be obtained with lstat(2)

changed.

FAMDeleted A file or directory being monitored was deleted.

FAMStartExecuting

A monitored, executable file started executing. The event occurs only the first time the file is executed.

FAMStopExecuting

A monitored, executable file that was running finished. If multiple processes from an executable are running, this event is generated only when the last one finishes.

FAMCreated A file was created in a directory being monitored.

FAMAcknowledge

FAM generates a FAMAcknowledge event in response to a call to FAMCancelMonitor(). If you specify an invalid request, that is, a relative path, FAM automatically cancels the request and immediately sends a FAMAcknowledge event.

FAMExists

When the application requests that a file be monitored, FAM generates a FAMExists event for that file (if it exists). When the application requests that a directory be monitored, FAM generates a FAMExists event for that directory (if it exists) and every file contained in that directory.

FAMEndExist

When the application requests a file or directory be monitored, FAM generates a FAMEndExist event after the last FAMExists event. (Therefore if you monitor a file, FAM generates a single FAMExists event followed by a FAMEndExist event.)

Note: Prior to IRIX 6.2, FAMNextEvent () did not initialize the *filename* field in a FAMEndExist event. You should use the request number to find the file or directory these events reference.

Symbolic Links

If you specify the pathname of a symbolic link to FAMMonitorDirectory() or FAMMonitorFile(), FAM monitors only the symbolic link itself, not the target of the link. Although it might seem logical to automatically monitor the target of a symbolic link, consider that if the target is on an automounted filesystem, monitoring the target

triggers and holds an automount. Another reason to monitor the link instead of the target is that the target may not exist.

There is no general solution for monitoring targets of symbolic links. You might decide that it's appropriate for your application to monitor a target even if it's automounted.

Tip: The libc routine realpath(3C) is useful when you need to resolve a link into its ultimate target.

Tip: Use statvfs(2) to recognize a remote file.

On the other hand, to avoid triggering and holding an automount, you can manually follow symbolic links until you reach either a local target, which you can then monitor, or a non-existent filesystem, in which case you might decide not to monitor the target. Another option is to test the target once to see if it is local, which triggers an automount only once if the target is automounted.

For example, the following routine determines if a given path is nonexistent, a dangling link, local, or remote.

```
#include <errno.h>
#include <limits.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/statvfs.h>

/*
   * determine a file's location
   */
enum location { ERROR, NONEXISTENT, DANGLING, LOCAL, REMOTE };
enum location
file_location(const char *path)
{
     char target_path[PATH_MAX];
     struct stat statbuf;
     struct statvfs svfsbuf;
     if (!realpath(path, target_path))
```

```
* realpath failed -- probably a permission
               problem, dangling link or nonexistent file.
            * /
            if (errno == EACCES)
                return ERROR;
            if (lstat(path, &statbuf) == 0)
                return DANGLING;
            else if (errno == ENOENT)
                return NONEXISTENT;
            else
                return ERROR;
      }
        Realpath succeeded. Find out if file is local.
      if (statvfs(target_path, &svfsbuf) < 0)</pre>
          return ERROR;
      if (svfsbuf.f_flag & ST_LOCAL)
          return LOCAL;
      else
          return REMOTE;
}
```

FAM Examples

The following examples show event streams that FAM sends in certain situations.

Example: A client monitors an existing file. Later, another program appends data to the file. Even later, the client cancels the monitoring request.

```
User calls FAMMonitorFile(... "/a/b/c" ...)

FAM events: FAMExists /a/b/c

FAMEndExist /a/b/c

Other program appends to file.

FAM event: FAMChanged /a/b/c

User calls FAMCancelMonitor(...)

FAM event: FAMAcknowledge /a/b/c
```

Example: A client monitors a directory containing two files. Later, another program creates a third file.

Example: A client monitors an executable file which is already running. Later, the program exits.

Example: A client makes an invalid request.

```
User calls FAMMonitorDirectory(... "relative/path" ...)
FAM event: FAMAcknowledge relative/path
```

Example: A client monitors a nonexistent file. Later, another program creates the file.

Example: A client monitors a directory containing some files. Another program deletes the directory, then creates a new file with the same name as the directory.

```
User calls
               FAMMonitorDirectory(... "/a/b" ...)
FAM events:
               FAMExists
                                /a/b
               FAMExists
                                file_one
               FAMExists
                                file_two
               FAMEndExist
                                /a/b
Directory and files are deleted.
FAM events:
               FAMDeleted
                                /a/b
               FAMChanged
                                /a/b
               FAMDeleted
                                file_one
```

FAMDeleted file_two

File with same name created.

FAM events: FAMCreated /a/b
FAMChanged /a/b

Using FAM

As noted in "Detecting Changes to Files and Directories" on page 112, your application can check for changes in files in directories that it monitors in two ways:

- use select() to wait until the FAM socket is active, indicating a change, which is described in "Waiting for File Changes."
- use FAMPending() to periodically poll FAM, which is explained in "Polling for File Changes."

This section describes how to use both approaches.

Waiting for File Changes

Follow these steps to use FAM in your application, using the select approach to detect changes:

- 1. Call FAMOpen() to create a connection to fam. This routine returns a FAMConnection structure used in all FAM procedures.
- 2. Call FAMMonitorFile() and FAMMonitorDirectory() to tell fam which files and directories to monitor.
- Select on the fam socket file descriptor and call FAMNextEvent() when the fam socket is readable.
- 4. When the application is finished monitoring a file or directory, call FAMCancelMonitor(). If you want to temporarily suspend monitoring of a file or directory, call FAMSuspendMonitor(). When you're ready to start monitoring again, call FAMResumeMonitor().
- 5. When the application no longer needs to monitor files and directories, call FAMClose() to release resources associated with files still being monitored and to close the connection to fam. This step is optional if you simply exit your application.

Example 8-1 demonstrates this process in a simple program.

Example 8-1 Using Select With FAM

```
monitor.c -- monitor arbitrary file or directory
                  using fam
 * /
#include <fam.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/select.h>
/* event_name() - return printable name of fam event code */
const char *event_name(int code)
    static const char *famevent[] = {
        "FAMChanged",
        "FAMDeleted",
        "FAMStartExecuting",
        "FAMStopExecuting",
        "FAMCreated",
        "FAMMoved",
        "FAMAcknowledge",
        "FAMExists",
        "FAMEndExist"
    };
    static char unknown_event[10];
    if (code < FAMChanged || code > FAMEndExist)
        sprintf(unknown_event, "unknown (%d)", code);
        return unknown_event;
    return famevent[code];
void main(int argc, char *argv[])
```

007-2006-130

```
{
    int i, nmon, rc, fam_fd;
    FAMConnection fc;
    FAMRequest *frp;
    struct stat status;
    FAMEvent fe;
    fd_set readfds;
    /* Allocate storage for requests */
    frp = malloc(argc * sizeof *frp);
    if (!frp)
        perror("malloc");
        exit(1);
    }
    /* Open fam connection */
    if ((FAMOpen(&fc)) < 0)
        perror("fam");
        exit(1);
    /* Request monitoring for each program argument */
    for (nmon = 0, i = 1; i < argc; i++)
        if (stat(argv[i], &status) < 0)</pre>
            perror(argv[i]);
            status.st_mode = 0;
        if ((status.st_mode & S_IFMT) == S_IFDIR)
            rc = FAMMonitorDirectory(&fc, argv[i], frp + i,
                                      NULL);
        else
            rc = FAMMonitorFile(&fc, argv[i], frp + i, NULL);
        if (rc < 0)
```

```
perror("FAMMonitor failed");
        continue;
    nmon++;
if (!nmon)
    fprintf(stderr, "Nothing monitored.\n");
    exit(1);
/* Initialize select data structure */
fam_fd = FAMCONNECTION_GETfd(&fc);
FD_ZERO(&readfds);
FD_SET(fam_fd, &readfds);
/* Loop forever. */
while(1)
    if (select(fam_fd + 1, &readfds,
               NULL, NULL, NULL) < 0)</pre>
         perror("select failed");
         exit(1);
    if (FD_ISSET(fam_fd, &readfds))
        if (FAMNextEvent(&fc, &fe) < 0)</pre>
            perror("FAMNextEvent");
            exit(1);
        printf("%-24s %s\n", fe.filename,
               event_name(fe.code));
```

007-2006-130

Polling for File Changes

Follow these steps to use FAM in your application, using the polling approach to detect changes:

- 1. Call FAMOpen() to create a connection to fam. This routine returns a FAMConnection structure used in all FAM procedures.
- 2. Call FAMMonitorFile() and FAMMonitorDirectory() to tell fam which files and directories to monitor.
- 3. Call FAMPending() to determine when there is a pending FAM event and then call FAMNextEvent() when an event is detected.
- 4. When the application is finished monitoring a file or directory, call FAMCancelMonitor(). If you want to temporarily suspend monitoring of a file or directory, call FAMSuspendMonitor(). When you're ready to start monitoring again, call FAMResumeMonitor().
- 5. When the application no longer needs to monitor files and directories, call FAMClose() to free resources associated with files still being monitored and to close the connection to fam. This step is optional if you simply exit your application.

For example, you could use the polling approach in the monitor.c program listed in Example 8-1 by deleting the code pertaining to the select data structure and replacing the while loop with the code shown in Example 8-2, which demonstrates this process in a simple program.

Example 8-2 Polling With FAM

```
while(1)
{
    rc = FAMPending(&fc);
    if (rc == 0)
        break;
    else if (rc == -1)
        perror("FAMPending");
    if (FAMNextEvent(&fc, &fe) < 0)
    {
        perror("FAMNextEvent");
        exit(1);
    }
    printf("%-24s %s\n", fe.filename,
        event_name(fe.code));</pre>
```

}

This is a particularly useful approach if you want to poll for changes from within an Xt work procedure. Example 8-3 shows the skeleton code for such a work procedure.

Example 8-3 Polling FAM Within an Xt Work Procedure

```
Boolean monitorFiles(XtPointer clientData)
{
   int rc = FAMPending(&fc);

   if (rc == 0)
       return(FALSE);
   else if (rc == -1)
       XtAppError(app_context, "FAMPending error");

   if (FAMNextEvent(&fc, &fe) < 0)
   {
       XtAppError(app_context, "FAMNextEvent error");
   }

   handleFileChange(fe);
   return(FALSE);
}</pre>
```

Providing Online Help With SGIHelp

This chapter describes how to use the Silicon Graphics online help system, SGIHelp, to deliver the online help for your product. It describes how to prepare the help and integrate it into your application. It contains the following sections:

- "Overview of SGIHelp" on page 126 provides an overview of the help system.
- "The SGIHelp Interface" on page 129 describes the SGIHelp API.
- "Implementing Help in an Application" on page 133 provides some examples of implementing online help in an application.
- "Application Helpmap Files" on page 138 describes the format and use of application helpmap files.
- "Writing the Online Help" on page 145 describes how to write the source files containing your application's online help.
- "Producing the Final Product" on page 150 describes how to compile your help files into viewable form and package them for installation on your users' systems.
- "Bibliography of SGML References" on page 152 list additional references.

The section "Online Help" in Chapter 4 of the *IRIX Interactive User Interface Guidelines* provides interface and content guidelines for adding online help to your application.

Overview of SGIHelp

The SGIHelp system consists of a help viewer, a help library and include file, help document files, and optional application helpmap files. This section describes:

- "The Help Viewer"
- "The SGIHelp Library and Include File"
- "Help Document Files"
- "Application Helpmap Files"

Note: To develop online help for your application, you must install the **insight_dev** product, which contains the SGIHelp library and include file, help generation tools, examples, and templates.

The Help Viewer

The SGIHelp viewer, sgihelp(1), also referred to as the **help server**, displays help text in easy-to-use browsing windows. Figure 9-1 shows an example of a help window.

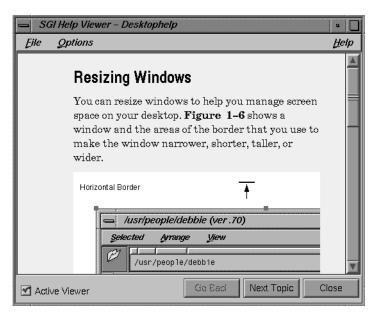


Figure 9-1 The Help Viewer

sgihelp can also display an index of all help topics available in a help document and allow the user to select a particular topic from the list. Figure 9-2 shows an example of a help index.

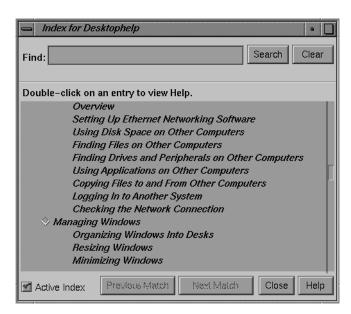


Figure 9-2 The Help Index Window

sgihelp is a separate application that gets started automatically whenever an application makes a help request. Neither users nor your application should ever need to explicitly start sgihelp. After the user closes all help windows, sgihelp remains running in the background for a few minutes. If it receives no other help requests within that time, it automatically exits.

The SGIHelp Library and Include File

The Silicon Graphics help library, libhelpmsg, handles communication with the help server. libhelpmsg depends on the libX11 library. Be sure to specify -lhelpmsg before -lX11 in the compilation or linking command.

For example, to compile a file hellohelp.c++ to produce the executable **hellohelp**, you would enter:

```
CC -o hellohelp hellohelp.c++ -lhelpmsg -lX11
```

You must include <helpapi/HelpBroker.h> in any source file that accesses online help. Both the library and include file were developed in C, and can be used with either the C or C++ programming languages.

Help Document Files

Help document files contain the actual help text in Standard Generalized Markup Language (SGML) format. In addition to text, help documents can contain graphics and hypertext links to other help topics.

Application Helpmap Files

Application helpmap files are optional; an application can request specific help topics directly. Applications helpmap files provide a level of indirection that allows you to structure your help presentation independently of your application code. The SGIHelp library also uses helpmaps to make it easier for you to implement context-sensitive help in your application.

Note: You must provide a helpmap for your application if you want a help index.

The SGIHelp Interface

This section describes the functions you use to access the help server from your application:

- "Initializing the Help Session"
- "Displaying a Help Topic"
- "Displaying the Help Index"

Initializing the Help Session

Before calling any other help functions, your application must first call SGIHelpInit():

```
int SGIHelpInit (Display *display, char *appClass, char *separator);
```

display The application's Display structure.

appClass The application's class name. Use the same name as you provide to

XtAppInitialize().

separator The separator character used by the application to separate the widget

hierarchy when a context-sensitive help request is made. At this time,

you must use the period (.).

SGIHelpInit() does not start or communicate with the help server process; it simply initializes data structures for the other SGIHelp functions. SGIHelpInit() returns 1 on success, and 0 on failure.

Example 9-1 shows an example of how to use SGIHelpInit().

```
Example 9-1 Initializing a Help Session Using SGIHelpInit()
```

```
#include <Xm/Xm.h>
#include <helpapi/HelpBroker.h>
void main ( int argc, char **argv )
{
                mainWindow; /* Main window shell widget */
                           /* An application context,
    XtAppContext app;
                            * needed by Xt
                           /* Return status */
    int
               status;
    /* ... */
   mainWindow = XtAppInitialize ( &app, "MyApp", NULL, 0,
                                  &argc, argv, NULL,
                                  NULL, 0 );
    /* Initialize the help session */
    status = SGIHelpInit( XtDisplay(mainWindow),
                          "MyApp", ".");
```

```
/* ... */
}
```

Displaying a Help Topic

To request display of a help topic from within your application, call SGIHelpMsg():

int SGIHelpMsg (char *key, char *book, char *userData);

key Specifies either 1) the ID of a particular help topic in a help document, or 2) a widget hierarchy.

If you provide a help ID, the help server displays the help topic identified in the help document specified by the *book* argument. You must provide a help book name in this case. See "Writing the Online Help" on page 145 for an explanation of help IDs.

If you provide a widget hierarchy, the help server looks in the application's helpmap file to find a mapping. If it doesn't find an exact match, it uses a fallback algorithm to determine which is the "closest" hierarchy found. Typically you use this technique to provide context-sensitive help. See "Application Helpmap Files" on page 138 for more information about the helpmap file.

book

Gives the **short name** of the help document containing the application's help information. See "Writing the Online Help" for a description of help document short names.

If you set this to NULL or asterisk (*), the help server looks in the application's helpmap file for the book name. In this case, a helpmap file must exist. See "Application Helpmap Files" for more information about the helpmap file.

userData

Reserved for future use. You should always set this field to NULL.

If a copy of the help server is not already running, SGIHelpMsg() automatically starts the server. SGIHelpMsg() returns 1 on success, and 0 on failure.

Example 9-2 shows an example of using SGIHelpMsg() to display the help topic identified by the help ID "help_save_button" in the help document with the short name "MyAppHelp."

Example 9-2 Requesting a Specific Help Topic Using SGIHelpMsg()

```
#include <helpapi/HelpBroker.h>
/* Assume initialization of help session is complete */
/*
   * This call displays the help topic with a key of
   * "help_save_button" (found in the "HelpId=" field).
   * It will look for this section in the help document
   * "MyAppHelp".
   */
status = SGIHelpMsg( "help_save_button", "MyAppHelp", NULL );
```

Example 9-3 shows an example of using SGIHelpMsg() to request help given a widget hierarchy. In this case, the application must have a helpmap file, and the help file must contain an entry mapping the given hierarchy to a help topic for this call to succeed.

Example 9-3 Requesting a Help Topic for a Widget Using SGIHelpMsg()

Displaying the Help Index

The SGIHelpIndexMsg() call causes the help server to look for the application's helpmap file and to display the Help Index window:

```
int SGIHelpIndexMsg (char *key, char *book);
key You should always set this field to NULL or "index."
book Reserved for future use. You should always set this field to NULL.
```

The index displays all the help topics in the helpmap file in the order they appear in the file. You must have a helpmap file for this call to work properly. See "Application Helpmap Files" on page 138 for more information about the helpmap file. SGIHelpIndexMsg() returns 1 on success, and 0 on failure.

Example 9-4 shows an example of how to use SGIHelpIndexMsg().

```
Example 9-4 Displaying a Help Index Using SGIHelpIndexMsg()
/* Assume initialization of help session is complete */
/*
    * This call will look in the application's helpmap
    * file for a list of topics to display to the user in
    * sgihelp's index window.
    */
status = SGIHelpIndexMsg( "index", NULL );
```

Implementing Help in an Application

The section "Types of Online Help" in Chapter 4 of the *IRIX Interactive User Interface Guidelines* describes the user interfaces to online help that your application should provide. In summary, these services are:

- Help menus in all application windows with menu bars
- Help buttons in all applications without menu bars
- Context-sensitive help available through both the help menus and the <shift+F1> keyboard accelerator.

This section contains specific suggestions for implementing these help interfaces to your application:

- "Constructing a Help Menu"
- "Implementing a Help Button"
- "Providing Context-Sensitive Help"

Constructing a Help Menu

For those windows in your application with a menu bar, you should provide a Help menu. "Providing Help through a Help Menu" in Chapter 4 of the *IRIX Interactive User Interface Guidelines* recommends that the following entries appear in the Help menu:

"Click for Help"

Provides context-sensitive help. This option should also use the <shift+f1> keyboard accelerator. When a user selects "Click for Help," the cursor should turn into a question mark (?). The user can then move the cursor over an item or area of interest and click. Your application should then display a help topic describing the purpose of the item or area.

"Providing Context-Sensitive Help" on page 136 provides detailed instructions for implementing context-sensitive help.

"Overview"

Displays overview information. The main primary window should provide an overview of the application. For other windows, this option should appear as "Overview for *<window name>*" and provide an overview of the current window only.

A list of topics and tasks

This section should contain a list of topics and tasks that the user can perform in your application. When the user selects one of the options, your application should display a help topic for that item. To reduce the size of this section, you can move some of the tasks to submenus.

You can hard code the entries in this section or, if you have a helpmap file for your application, you can parse the helpmap and dynamically create the task and subtask entries.

"Index"

Displays Help Index window for the application. You must have an application helpmap file to support this option.

"Keys & Shortcuts"

Displays the application's accelerator keys, keyboard shortcuts, and other actions in the application.

"Product Information"

Displays a dialog box showing the name, version, and any copyright information or other related data for your application. Typically, you should present this information using an IRIS IM dialog rather than using online help.

Separators are added automatically. See the program listing in Example C-4 for an example of creating a Help menu.

Implementing a Help Button

For those windows in your application that don't contain a menu bar, you should provide a **Help** button. Example 9-5 shows how you can use the SGIHelp API to communicate with the help server from a pushbutton within your application. "Providing Help Through a Help Button" in Chapter 4 of the *IRIX Interactive User Interface Guidelines* provides guidelines for when to implement a **Help** button.

Example 9-5 Providing a **Help** Button

```
/* required include file for direct communication with help server */
#include <helpapi/HelpBroker.h>
#include <Xm/Xm.h>
/* ... */
/* initialize help server information */
SGIHelpInit(display, "MyWindowApp", ".");
. . .
/* create help pushbutton for your window */
Widget helpB = XmCreatePushButton(parent, "helpB", NULL, 0);
XtManageChild(helpB);
 XtAddCallback(helpB, XmNactivateCallback,
                (XtCallbackProc)helpCB, (XtPointer)NULL);
/* ... */
/* help callback */
void helpCB(Widget w, XtPointer clientData, XtPointer callData)
                 * communicate with the help server; developer
                 * may wish to pass the "key" in as part of the
                 * callback's callData parameter...
                SGIHelpMsg("key", "book", NULL);
```

Providing Context-Sensitive Help

To provide context-sensitive help from within your application, you need to write code that tracks the cursor and interrogates the widget hierarchy. Additionally, you need to make a mapping between what the user has clicked, and the help card that's displayed.

The best way to provide the mapping is with the application helpmap file. The SGIHelp library provides a fallback algorithm for finding help topics that simplifies the process mapping widgets to topics. If the help system can't find an exact match to the widget string in the helpmap file, it drops the last widget from the string and tries again. The help system reiterates this process until it finds a match in the helpmap file. This eliminates the need to explicitly map a help topic for every widget in your application. Instead you can map a help topic to a higher-level manager widget and have that topic mapped to all of its descendent widgets as well.

For more information on the structure of application helpmap files, see "Application Helpmap Files" on page 138.

Example 9-6 shows the code used to implement context-sensitive help in the example program listed in Example C-4, which simply installs <code>clickForHelpCB()</code> as the callback function for the "Click for Help" option of the Help menu. As long as you create a helpmap file for your application, you can use this routine as listed in your application as well.

Example 9-6 Implementing Context-Sensitive Help

```
void clickForHelpCB(Widget wid, XtPointer clientData, XtPointer callData)
{
    static Cursor cursor = NULL;
    static char path[512], tmp[512];
    Widget shell, result, w;

    strcpy(path, "");
    strcpy(tmp, "");

/*
    * create a question-mark cursor
    */
    if(!cursor)
        cursor = XCreateFontCursor(XtDisplay(wid), XC_question_arrow);

XmUpdateDisplay(_mainWindow);
```

```
/*
* get the top-level shell for the window
    shell = _mainWindow;
    while (shell && !XtIsShell(shell)) {
           shell = XtParent(shell);
* modal interface for selection of a component;
* returns the widget or gadget that contains the pointer
    result = XmTrackingLocate(shell, cursor, FALSE);
    if( result ) {
        w = result;
* get the widget hierarchy; separate with a '.';
* this also puts them in top-down vs. bottom-up order.
* /
        do {
              if( XtName(w) ) {
                  strcpy(path, XtName(w));
                  if( strlen(tmp) > 0 ) {
                      strcat(path, ".");
                      strcat(path, tmp);
                  strcpy(tmp, path);
              }
              w = XtParent(w);
         } while (w != NULL && w != shell);
          * send msg to the help server-widget hierarchy;
                 OR
          * provide a mapping to produce the key to be used
          * In this case, we'll let the sgihelp process do
          * the mapping for us, with the use of a helpmap file
          * Note that parameter 2, the book name, can be found
```

```
* from the helpmap file as well. The developer need
* not hard-code it, if a helpmap file is present for
* the application.
*

*/
if( strlen(path) > 0 ) {
        SGIHelpMsg(path, NULL, NULL);
}
}
```

Application Helpmap Files

Application helpmap files provide a level of indirection that allows you to structure your help presentation independently of your application code.

- "Helpmap File Conventions"
- "Helpmap File Format"
- "Widget Hierarchies in the Helpmap File"

You don't have to create a helpmap for your application, but doing so gives you the following benefits:

- Your application can display a Help Index window, allowing users to select a particular topic directly from the list.
- You can write the code that generates your application's Help menu to create the
 "list of topics and tasks" options dynamically from the helpmap. You can then add
 and restructure your task help without recompiling your application. See
 "Constructing a Help Menu" on page 134 for details on the Help menu's list of
 topics.
- Your application's Help menu can launch a browser and access a URL on the World Wide Web. See "Example of Helpmap Entry to Access a Web Browser" for more information.
- You can provide context-sensitive without hard-coding in your source code a help topic to each widget. The SGIHelp library provides a fallback algorithm for finding help topics that simplifies the process mapping widgets to topics. If the help system can't find an exact match to the widget string in the helpmap file, it drops the last widget from the string and tries again. The help system reiterates this process until it finds a match in the helpmap file. This eliminates the need to explicitly map a help

topic for every widget in your application. Instead you can map a help topic to a higher-level manager widget and have that topic mapped to all of its descendent widgets as well. See "Providing Context-Sensitive Help" on page 136 for information on implementing context-sensitive help in your application.

Helpmap File Conventions

Helpmap files are ASCII text files. The name of your application helpmap file must be "appClass.helpmap", where appClass is your application's class name as provided in your application's call to SGIHelpInit(). See "Initializing the Help Session" on page 130 for more information on SGIHelpInit().

If you create a helpmap file for your application, you must create a subdirectory named help in the directory containing your help document and put all of your document's figures in that subdirectory. See "Preparing to Build the Online Help" on page 147 for more information.

Helpmap File Format

Each entry, or **help topic**, in a helpmap consists of a single line containing at least six fields, each field separated by semicolons:

type; book; title; level; helpID; widget-hierarchy[; widget-hierarchy ...]

Helpmap Fields

All fields are required for each entry. Their purpose is as follows:

type The type of help topic. Its value can be:

- 0 A context-sensitive topic.
- 1 The overview topic.
- A task-oriented entry that could show up in the "list of topics and tasks" area of the Help menu. See "Constructing a Help Menu" on page 134 for details on the Help menu's list of topics.
- 3 The Keys and Shortcuts topic.

book The name of the help document that contains this help topic. Help topics

can reside in different books. Each individual help topic can point to

only one help book.

title The title of the help topic. This appears in the Help Index window. If

your application parses the helpmap file to generate the "list of topics and tasks" area of the Help menu, you can use this as the label for the

menu option.

level A number determining the topic level. A value of 0 indicates a main

topic, a value of 1 a sub-topic, a value of 2 a sub-sub-topic, and so forth. This produces an expandable/collapsible outline of topics for the Help

Index window.

If your application parses the helpmap file to generate the "list of topics and tasks" area of the Help menu, you can also use these values to

construct "roll-over" submenus as part of a Help menu.

helpID The unique ID, as specified by the "HelpID" attribute, of the specific

help topic in the help document.

widget-hierarchy

One or more fully-qualified widget specifications for use with context-sensitive help. You can provide multiple specifications, delimited by semicolons, to associate different areas with the same

topics.

Examples of Helpmap Entries

For example, the following entry in Swpkg.helpmap specifies the overview topic:

1;IndigoMagic_IG;Overview;O;Overview;Swpkg.swpkg.overview

The following entries from Swpkg. helpmap specify several context-sensitive help topics. In this case, the first entry appears as a main topic in the Help Index window and the next three appear as sub-topics:

```
0;Swpkg_UG;Using the swpkg Menus;0;menu.bar;Swpkg.swpkg.menuBar
0;Swpkg_UG;The File Menu;1;menu.bar.file;Swpkg.swpkg.menuBar.File
0;Swpkg_UG;The View menu;1;menu.bar.view;Swpkg.swpkg.menuBar.View
0;Swpkg_UG;The Help menu;1;menu.bar.help;Swpkg.swpkg.menuBar.helpMenu
```

The following shows a more complex hierarchy from Swpkg.helpmap:

2;Swpkg_UG;Tagging Files;0;tag.files.worksheet;Swpkg.swpkg

- 2;Swpkg_UG;Selecting Product Files;1;file.browser;Swpkg.swpkg.view.viewPanedWindow.viewForm.\leftForm.filesBody.addBody.FileListAdd.selectionGrid
- 0;Swpkg_UG;Setting the Browsing Directory;2;file.browser.dirfield;Swpkg.swpkg.view.\viewPanedWindow.viewForm.leftForm.filesBody.addBody.FileListAdd.directoryLabel;Swpkg.swpkg.\view.viewPanedWindow.viewForm.leftForm.filesBody.addBody.FileListAdd.directoryTextField
- 0;Swpkg_UG;Selecting Files From the File List;2;file.browser.filelist;Swpkg.swpkg.view.\viewPanedWindow.viewForm.leftForm.filesBody.addBody.FileListAdd.scrolledWindow.filesList;\Swpkg.swpkg.view.viewPanedWindow.viewForm.leftForm.filesBody.addBody.FileListAdd.\scrolledWindow.VertScrollBar

Note: The backslashes (\) indicate linewraps; they do not actually appear in the helpmap file. Each helpmap entry must be a single line.

In the example above, the first entry is a task-oriented topic (2 in the *type* field). swpkg parses the helpmap file to create its Help menu, so "Tagging Files" appears as a selection. The second entry is also a task-oriented topic. It's a sub-topic of the first entry and appears in a submenu off the "Tagging Files" selection. The last two entries are marked as context-sensitive only (0 in the *type* field). These entries don't appear anywhere in the application's Help menu, but they do appear as sub-sub-topics in the Help Index window. Also note that the last two entries have two widget specifications, providing context-sensitive help for two different widgets.

Note: The order of the entries in the application helpmap file determines the order in which help topics appear in the Help Index window.

Example of Helpmap Entry to Access a Web Browser

You can put an entry into a helpmap file to launch a Web browser and access a URL. An example entry in the showcase helpmap file looks like this:

2;HREF=http://www.sgi.com/Products/SGIHelp_Hub/Showcase_3.3.2.html;\
Showcase Web Page;0;showcase_web;Showcase3D.showcase2;ShowcaseUG;\
About Gizmos;0;about_gizmo;Showcase3D.showcase

Note: The backslashes (\) indicate linewraps; do not actually enter them in the helpmap file. Each helpmap entry must be a single line.

When SGIhelp encounters this type of entry, it uses a Web browser such as Netscape (or \$WEBBROWSER) to show the URL specified by the entry.

Widget Hierarchies in the Helpmap File

At least one widget hierarchy must accompany every point in the application helpmap file. That one (default) point should be set to "application_classname.top-level_shell".

Note that the application class name must always be the first component of a widget hierarchy string. All widget ID's within the string must be delimited by a period (.).

Widget hierarchies can be as fine-grained as you wish to make them. A fall-back algorithm is in place (to go to the closest available entry) when the user clicks a widget in context-sensitive help mode. For example, suppose your application includes a row or set of buttons. When the user asks for help on a button, you pass that widget string to SGIHelp. If the widget string is not found in the mappings, the last widget is dropped off the string (in this case, the widget ID for the button itself). The new string is compared to all available mappings. This loop continues until something is found. At the very least, you should fall back to an "Overview" card.

To get a sample widget hierarchy (help message) from an application, you can run the SGIHelp help server process in debug mode. Before doing this, you need to add the SGIHelp API call, SGIHelpMsg(), to your application and implement context-sensitive help. Make sure that you send a widget hierarchy string for the "key" parameter in the SGIHelpMsg() call. (See "Providing Context-Sensitive Help within an Application" and "Understanding Available Calls" for details on this call.)

To get a sample widget hierarchy from an application that implements context-sensitive help, follow these steps:

- 1. Bring up a shell.
- 2. Make sure the help server process isn't running. Type:
 - % /etc/killall sgihelp
- 3. Type the following to make the help server process run in the foreground in debug mode:
 - % /usr/sbin/sgihelp -f -debug
- 4. Run your application, and then choose "Click for Help" from the help menu. The cursor should change into a question mark (?), or whatever cursor you've implemented for context sensitive help.
- 5. Click a widget or an area of the application.
- 6. Check the shell from which SGIHelp is being run. You should see a line such as:

```
REQUEST= client="Overview" command="view" book=""
keyvalue="DesksOverview.MainView.Frame.viewport.Bboard"
separator="." user_data=""
```

The "keyvalue" field contains the widget hierarchy that you can add to the helpmap file. Remember to add the application class name to the front of the string. For the example above, the full widget hierarchy string would be:

Overview.DesksOverview.MainView.Frame.viewport.Bboard

Writing the Online Help

This section describes how you prepare the online help document. It provides an explanation of the standard format you must use, as well as the steps you take to actually prepare the file. Topics include:

- "Overview of Help Document Files"
- "Viewing the Sample Help Document Files"
- "Creating a Help Document File"
- "Preparing to Build the Online Help"
- "Building the Online Help"
- "Finding and Correcting Build Errors"

For guidelines on structuring and writing your online help text, see "Writing Online Help Content for SGIHelp" in Chapter 4 of the *IRIX Interactive User Interface Guidelines*.

Overview of Help Document Files

Help document files contain the actual help text in **Standard Generalized Markup Language** (SGML) format. When you write the online help for your product, you need to embed SGML tags to describe the structure of your document.

The file /usr/share/Insight/XHELP/samples/sampleDoc/sample.sgm is an example of a file with embedded SGML tags. (Example C-1 also lists this file.) Notice the tags surrounded by angle brackets (<>). These tags describe how each item fits into the structure of the overall document. For example, a paragraph might be tagged as a list item, and a word within that paragraph may be tagged as a command.

The **Document Type Definition** (DTD) outlines the tagging rules for your online documentation. In other words, it specifies which SGML tags are allowed, and in what combination or sequence. The file /usr/share/Insight/XHELP/dtd/XHELP.dtd lists the legal structure for your online help.

A DTD can be difficult to read, so you might instead want to look at the file /usr/share/Insight/XHELP/samples/XHELP_elements/XHELP_elements.s gm, which lists the legal elements in a help document and describes when to use them in your documents. (Example C-2 also lists this file.)

For a more complete understanding of SGML, refer to the bibliography in "Bibliography of SGML References" on page 152. It lists several of the many books on SGML.

Viewing the Sample Help Document Files

Before beginning to write your own help documents, you might find it helpful to examine the source of the sample help documents and then view resulting online versions. You can compile and view the help documents in Insight. To do so, follow these steps:

- 1. Go to a directory in which you want to build the sample help book.
- 2. Copy the necessary directories and files by entering:

```
% cp -r /usr/share/Insight/XHELP/samples .
```

3. Enter:

```
% cd samples/sampleDoc
```

- 4. Build the file sample.sgm by entering:
 - % make help
- 5. To view this file, enter:

```
% iiv -b . -v sample
```

6. Change to the exampleApp directory by entering:

```
% cd ../exampleApp
```

7. Build the file exampleAppXmHelp.sgm by entering:

```
% make help
```

8. To view this file, enter:

```
% iiv -b . -v exampleAppXmHelp
```

Creating a Help Document File

To create the help document file for your application:

- 1. Create a new directory for the online help, then go to this directory.
- 2. Create a text file and name the file "title.sgm", where *title* is one word that identifies the online help.

3. Write the online help.

You can include figures as described in the example help documents. If your document contains figures, create a subdirectory named either figures or online in your help document directory and put all of your document's figures in that subdirectory.

Preparing to Build the Online Help

After writing your online help you must **build** it, similarly to the way you compile a program. When you build the online help, you transform the raw SGML file into a viewable, online document. To get started, you need to create two files: a Makefile and a spec file.

The Makefile specifies:

- the name of file that contains the online help
- the name you want to assign to the help book
- the version number of the product

The spec file specifies:

- the title of your product
- the official release and version numbers
- other information that is used when you create the final, installable images

To create these files, follow these steps:

- 1. Go to the directory that contains the online help file.
- 2. Copy /usr/share/Insight/XHELP/templates/Makefile_xhelp by typing:
- cp /usr/share/Insight/XHELP/templates/Makefile_xhelp Makefile
- Copy /usr/share/Insight/XHELP/templates/spec_xhelp by typing:
- cp /usr/share/Insight/XHELP/templates/spec_xhelp spec

4. Edit the Makefile:

- Next to the label TITLE, type the name of the file that contains the online help.
- Next to the label FULL_TITLE, type the name you want to assign to the help book. This name can contain several words, and is used only if you decide to display the help as a "book" on the Insight bookshelf.
- Next to the label VERSION, type the version number for the product.
- Next to the label HIDDEN, remove the comment character (#) if you want the online help to appear as a book on an Insight bookshelf. Change this if you want users to be able to browse the help information using Insight, and not just from within your application.

5. Edit the spec file:

- Replace the string \${RELEASE} with the release number for the product. This should match what you've entered in the Makefile for the VERSION.
- Replace the string <ProductName> with a one-word name for the product.
- Replace the string <Shortname> with the TITLE you specified in the Makefile.
- Replace the string <SHORTNAME> with the TITLE you specified in the Makefile. Capitalize all letters.
- Replace the string <SHORTNAME_HELP> with the TITLE followed by "HELP".
- Replace the string <Book title> with the FULL_TITLE you specified in the Makefile.

Once you have edited these files, the directory containing your help document should contain:

- your help document
- the Makefile
- the spec file
- if you included figures in your help document, a subdirectory named either figures or online containing all of the figures
- if you created a helpmap file for you application, a subdirectory named help containing the helpmap file

Building the Online Help

Once you have written the online help and done the preparation described in "Preparing to Build the Online Help" on page 147, you can build and view the online help. To do so, follow these steps:

- 1. Go to the directory that contains the online help files.
- 2. Enter:

```
% make help
```

If the help is formatted properly, the online help will build. You should see a file called booklist.txt and a directory called books.

If the SGML file contains errors, you will see them displayed in the shell window. See "Finding and Correcting Build Errors" for details.

3. View the book by typing

```
% iiv -b . -v title
```

Where *title* is the value of TITLE from the Makefile.

Finding and Correcting Build Errors

The SGML tags come in pairs. Each pair contains an opening tag and a closing tag, and the tag applies to everything between the opening tag and the closing tag. If you use these tags incorrectly, you'll get error messages when you build the help file. The most common errors are the result of misspelled tag names, mismatched end tags, or tags used out of sequence.

Some examples of common error messages are:

```
mkhelperror: not authorized to add tag 'PAR', ignoring content.
```

This error appears if you specify an invalid tag. In this case, the invalid tag is "PAR." The valid tag name is "PARA."

This error message occurs when the parser sees a tag it isn't expecting. In this case it found a HELPLABEL that was not preceded by a HELPTOPIC start tag. The error message specifies the line number of the error (37), the context in the file, and the Fully Qualified Generic Identifier (FQGI) of the context. You can probably ignore the FQGI; it describes where the error occurs within the SGML structure.

This message can occur if you close items with the generic end tag, </>. In this case, the </> closes the <warning> because the start tag for <para> is missing. This may occur if you leave out a start tag or accidentally spell it incorrectly.

If you want additional information about the errors, use the command make verify. It produces a more detailed error log.

Producing the Final Product

This section describes how to package your online help as a subsystem that users can install using Software Manager (swmgr), the Silicon Graphics software installation utility. Topics include:

- "Creating the Installable Subsystem"
- "Incorporating the Help Subsystem into a Product With a Custom Installation Script"
- "Incorporating the Help Subsystem into an Installable Product"

Creating the Installable Subsystem

After you've finished writing and building your online help, you need to package it so that users can install it with the rest of your product. To do so:

- 1. Go to the directory that contains the online help.
- 2. Enter:

% make images

This produces a directory called images. This directory contains all of the files you need to let users install the online help using Software Manager.

Incorporating the Help Subsystem into an Installable Product

If you use the Software Packager utility (swpkg) to package your product so that users can install it using Software Manager, you need to merge the online help subsystem with the rest of your product. Consult the *Software Packager User's Guide* for detailed instructions for using swpkg.

You don't need to use swpkg to create spec or IDB files for your online help subsystem. By following the instructions in "Preparing to Build the Online Help" on page 147, you created the spec file. The process of building your online help, described in "Building the Online Help" on page 149 automatically created an IDB file and tagged the files; set the permissions and destinations; and assigned the necessary attributes. The online help build tools use "/" as the Source and Destination Tree Root directories when generating the IDB file. (The *Software Packager User's Guide* defines all of these terms.)

If you've not already created the spec and IDB files for the rest of your product using swpkg, you can use swpkg to open the existing help subsystem spec and IDB files, and expand them as needed to handle the rest of your product. Consult the *Software Packager User's Guide* for instructions.

If you've already created the spec and IDB files for your product, you can merge the help subsystem with the existing files as described in "Combining Existing Products Into a Single Product" in Chapter 7 of the *Software Packager User's Guide*.

Incorporating the Help Subsystem into a Product With a Custom Installation Script

If you don't use swpkg to package your product for installation with Software Manager, do one of the following.

• If users install your product using the tar command, have them use tar to copy the online help images as well. After copying the images, the user needs to type:

```
# inst -af <inst_product>
```

where *inst_product* is the location of the images.

• If you've created a script, enhance the script so that it extracts all of the help images onto disk, and then invokes the command:

```
# inst -af <inst_product>
```

where *inst_product* is the location of the images.

Bibliography of SGML References

- *SoftQuad, Inc. The SGML Primer. SoftQuad's Quick Reference Guide to the Essentials of the Standard: The SGML Needed for Reading a DTD and Marked-Up Documents and Discussing Them Reasonably. Version 2.0. Toronto: SoftQuad Inc., May 1991. 36 pages. Available from SoftQuad Inc.; 56 Aberfoyle Crescent, Suite 810; Toronto, Ontario; Canada M8X 2W4; TEL: +1 (416) 239-4801; FAX: +1 (416) 239-7105.
- 2. Bryan, Martin. *SGML: An Author's Guide to the Standard Generalized Markup Language*. Wokingham/Reading/New York: Addison-Wesley, 1988. ISBN: 0-201-17535-5 (pbk); LC CALL NO: QA76.73.S44 B79 1988. 380 pages. A highly detailed and useful manual explaining and illustrating features of ISO 8879. The book: (1) shows how to analyze the inherent structure of a document; (2) illustrates a wide variety of markup tags; (3) shows how to design your own tag set; (4) is copiously illustrated with practical examples; (5) covers the full range of SGML features. Technical and non-technical authors, publishers, typesetters and users of desktop publishing systems will find this book a valuable tutorial on the use of SGML and a comprehensive reference to the standard. It assumes no prior knowledge of computing or typography on the part of its readers.

- 3. Goldfarb, Charles F. *The SGML Handbook*. Edited and with a foreword by Yuri Rubinsky. Oxford: Oxford University Press, 1990. ISBN: 0-19-853737-1. 688 pages. This volume contains the full annotated text of ISO 8879 (with amendments), authored by IBM Senior Systems Analyst and acknowledged "father of SGML," Charles Goldfarb. The book was itself produced from SGML input using a DTD which is a variation of the "ISO.general" sample DTD included in the annexes to ISO 8879. The SGML Handbook includes: (1) the up-to-date amended full text of ISO 8879, extensively annotated, cross-referenced, and indexed; (2) a detailed structured overview of SGML, covering every concept; (3) additional tutorial and reference material; and (4) a unique "push- button access system" that provides paper hypertext links between the standard, annotations, overview, and tutorials.
- 4. Herwijnen, Eric van. *Practical SGML*. Dordrecht/Hingham, MA: Wolters Kluwer Academic Publishers. 200 pages. ISBN: 0-7923- 0635-X. The book is designed as a "practical SGML survival-kit for SGML users (especially authors) rather than developers," and itself constitutes an experiment in SGML publishing. The book provides a practical and painless introduction to the essentials of SGML, and an overview of some SGML applications. See the reviews by (1) Carol Van Ess-Dykema in Computational Linguistics 17/1 (March 1991) 110-116, and (2) Deborah A. Lapeyre in *<TAG>* 16 (October 1990) 12-14.
- 5. Smith, Joan M.; Stutely, Robert S. SGML: The Users' Guide to ISO 8879. Chichester/New York: Ellis Horwood/Halsted, 1988. 173 pages. ISBN: 0-7458-0221-4 (Ellis Horwood) and ISBN: 0-470-21126-1 (Halsted). LC CALL NO: QA76.73.S44 S44 1988. The book (1) supplies a list of some 200 syntax productions, in numerical and alphabetical sequence; (2) gives a combined abbreviation list; (3) includes highly useful subject indices to ISO 8879 and its annexes; (4) supplies graphic representations for the ISO 8879 character entities; and (5) lists SGML keywords and reserved names. An overview of the book may be found in the SGML Users' Group Newsletter 9 (August 1988).
- 6. ISO 8879:1986. Information Processing—Text and Office System—Standard Generalized Markup Language (SGML). International Organization for Standardization. Ref. No. ISO 8879:1986 (E). Geneva/New York, 1986. A subset of SGML became a US FIPS (Federal Information Processing Standard) in 1988. The British Standards Institution adopted SGML as a national standard (BS 6868) in 1987, and in 1989 SGML was adopted by the CEN/CENELEC Standards Committees as a European standard, #28879. Australia has dual numbered versions of ISO 8879 SGML and ISO 9069 SDIF (AS 3514—SGML 1987; AS 3649—1990 SDIF).
- 7. ISO 8879:1986 / A1:1988 (E). *Information Processing—Text and Office Systems— Standard Generalized Markup Language (SGML), Amendment 1.* Published 1988-07-01.
 Geneva: International Organization for Standardization, 1988.

Handling Users' System Preferences

This chapter describes how your application can recognize and use various system preferences that users can set through Desktop control panels. Whenever possible, your application should follow these preferences to provide a consistent interface for your users. In particular, this chapter contains:

- "Handling the Mouse Double-Click Speed Setting" describes how to recognize the preferred mouse double-click speed.
- "Using the Default Viewer and Editor Utilities Panel" describes the **dtUtilities** panel, lists utility variables and their values, and provides an example of how to set the preferred visual text editor when your application needs to let users edit text.

Handling the Mouse Double-Click Speed Setting

The Mouse Settings control panel (available from the "Customize" submenu of the Desktop toolchest) allows users to set various parameters that affect the operation of the mouse. The setting of importance to applications is "Click Speed," which determines the maximum interval between double-clicks. "Click Speed" sets the *multiClickTime X resource.

In most cases, you don't need to do anything to handle this setting. IRIS IM widgets automatically use the **multiClickTime** value as appropriate. Only if your application needs to handle double-clicks explicitly (for example, to select a word in a word processing application) does it need to call XtGetMultiClickTime() to determine the double-click time. See the XtGetMultiClickTime(3Xt) reference page for more information on XtGetMultiClickTime().

Note: Don't call XtSetMultiClickTime(), which sets the double-click time for the entire display.

Using the Default Viewer and Editor Utilities Panel

Users who select Toolchest->Customize->Utilities invoke /usr/sbin/dtUtilities. This application displays the "Default Viewer and Editor Utilities" panel (shown in Figure 10-1).



Figure 10-1 The dtUtilities Panel

Using the **dtUtilities** panel, users can select installed utilities, including the following:

- Text Editor (for example, xwsh, Jot, or Vi)
- Web Browser (for example, Netscape)
- Image Viewer (for example, Imgview)
- Mailer (for example, MediaMail or Netscape)
- Book Viewer (for example, Insight)
- PostScript Viewer (for example, ShowPS)

Selecting Utilities and Their Values

The **dtUtilities** variables and their values are listed in Table 10-1. An example of providing programmatic access to one of these utilities is in "Setting the Preferred Text Editor," which is below the table.

Table 10-1 dtUtilities and Their Values

dtUtility Variable	Value
WINEDITOR	Path to a window editor (for example, /usr/sbin/jot).
WEBBROWSER	Path to an X Window System application that is a World Wide Web visual browser (for example, /usr/bin/X11/netscape).
IMGVIEWER	Path to an application that displays image files (for example, /usr/bin/X11/imgview).
MAILBOXPROG	Path to a mail reader application (for example, /usr/bin/X11/MediaMail -gui).
BOOKVIEWER	Path to an application for displaying InSight books (for example, /usr/sbin/insight).
PSVIEWER	Path to an application for viewing PostScript files (for example, /usr/bin/X11/showps).

Setting the Preferred Text Editor

The "Text Editor" setting on the **dtUtilities** control panel allows users to select a preferred visual editor for editing ASCII text. This setting sets the value of the WINEDITOR environment variable.

The following instructions explain how to set the value of WINEDITOR. You can use the same instructions for the other utilities on the control panel. Just refer to Table 10-1 for the utility and value you wish to set.

Whenever your application needs to let users edit text, you should:

- 1. Call getenv() to check whether the WINEDITOR environment variable is set. See the getenv(3c) reference page for more information on getenv().
- 2. If WINEDITOR is set, save the text to edit in a temporary file. Typically, you should check the value of the environment variable TMPDIR and, if it is set, put the temporary file in that directory.
- 3. Execute the editor, providing it the name of the temporary file as an argument.
- 4. When the user quits the editor, read the temporary file and delete it.

Chapter 11:

Creating Desktop Icons: An Overview

Chapter 12:

Using IconSmith

Chapter 13:

File Typing Rules

Chapter 14:

Printing From the Desktop

Creating Desktop Icons: An Overview

This chapter offers an overview of the basic steps for creating IRIX Interactive Desktop icons and adding them to the Icon Catalog. If you don't feel you need much background information, you can skip to the brief list of instructions provided in "Checklist for Creating an Icon" on page 163.

This chapter contains these sections:

- "About IRIX Interactive Desktop Icons" on page 162 briefly discusses the IRIX Interactive Desktop and lists what kinds of icons you'll need to provide for your application.
- "Checklist for Creating an Icon" on page 163 lists the basic steps for drawing, programming, compiling, and installing an icon.
- "Creating an Icon: The Basic Steps Explained in Detail" on page 164 explains each of the basic icon creation steps in more detail.

Note: Minimized windows, which represent running applications, aren't Desktop icons. To learn how to customize the image on a minimized window, refer to Chapter 6, "Customizing Your Application's Minimized Windows."

About IRIX Interactive Desktop Icons

Files on the Desktop are represented by icons. Users can manipulate these icons to run applications, print documents, and perform other actions. "How Users Interact With Desktop Icons" in Chapter 1 of the *IRIX Interactive User Interface Guidelines* describes some of the common user interactions.

The Desktop displays different icons to represent the different types of files. For example, the default icon for binary executables is the "magic carpet," and the default icon for plain text files is a stack of pages.

When you create your own application, by default the Desktop uses an appropriate "generic" icon to represent the application and its associated data files (for example, the magic carpet icon for the executable and the stack of pages icon for text files). You can also design your own custom icons to promote product identity and to indicate associated files. For example, the custom ShowCase icons over a generic.exec.closed.fti and a generic.exec.open.fti look like this:





showcase

showcase

Another advantage of creating custom icons is that you can **program** them to perform certain actions when users interact with them on the Desktop. For example, you can program a custom data file icon so that when a user opens it, the Desktop launches your application and opens the data file.

The Desktop determines which icon to display for a particular file by finding a matching **file type**. A file type consists of a set of **File Typing Rules** (FTRs) that describe which files belong to the file type and how that type's icon looks and acts on the Desktop.

The Desktop reads FTRs from compiled versions of special text files called **FTR files**. An FTR file is a file in which one or more file types are defined (typically, you define more than one file type in a single file). FTR files can also contain **print conversion rules**, which define any special filters needed to print given file types. Chapter 13, "File Typing Rules," discusses the syntax of FTRs, and Chapter 14, "Printing From the Desktop," discusses print conversion rules.

Checklist for Creating an Icon

To provide a comprehensive Desktop icon interface for your application:

- Tag your application. You need to tag the application with its own unique identification number so that the Desktop has a way of matching the application with the corresponding FTRs. See "Step One: Tagging Your Application" on page 165 for instructions.
- 2. Draw a picture of your icon. Create a distinctive Desktop icon to help users distinguish your application from other applications on the Desktop. Optionally, create an icon for the data files associated with your application. Use the IconSmith application to draw your icons. IconSmith allows you to draw an icon and then convert it into the icon description language used by the Desktop. IconSmith is the only tool you can use to create an icon picture. For guidelines on designing icons, see the IRIX Interactive User Interface Guidelines. For information on how to use IconSmith, see Chapter 12, "Using IconSmith."
- 3. **Program your icon.** Create the FTRs to define your icons' Desktop interaction. Chapter 13, "File Typing Rules," describes FTRs in detail. Before programming your icon, think about what users expect from the application and, with that in mind, decide how you want the icon to behave within the Desktop. Before you make these decisions, read the icon programming guidelines in "Defining the Behavior of Icons With FTRs" in Chapter 2 of the *IRIX Interactive User Interface Guidelines*. In particular:
 - Program your Desktop icon to run your application with the most useful options. Include instructions for launching your application when the user opens the icon; opens the icon while holding down the <alt>key; and drags and drops other icons on the application icon.
 - If there are several useful combinations of options that users may want to use when invoking your application, you can incorporate them into a Desktop menu. (These Desktop menu items appear only when the icon is selected.) Users can then select the menu item that corresponds to the behavior they want—without having to memorize a lot of option flags.
 - Where appropriate, provide **print conversion rules** that describe how to convert a data file for printing into a type recognized by the Desktop. To print output, users can then just select the appropriate data file icon and choose "Print" from the Desktop menu rather than having to remember specialized filter information. Chapter 14, "Printing From the Desktop," describes print conversion rules.

- 4. **Compile the source files.** Compile the .ftr files into an .otr file. In particular, the desktop.otr file contains the compiled source for existing FTRs. For more information on .otr files, see "Step Four: Compiling the Source Files" on page 172.
- 5. Add your application to the Icon Catalog. This makes it easier for your users to locate your icon in the Icon Catalog and helps maintain a consistent look for your application in the Desktop. "Step Five: Installing Your Application in the Icon Catalog" on page 173 explains how to do this.
- 6. **Restart the Desktop.** You can view your changes after you restart the Desktop. "Step Six: Restarting the Desktop" on page 174 explains how to restart the Desktop.
- 7. **Update your installation process.** If you want to install your application on other Silicon Graphics workstations, include in your installation all of the files that you created in the preceding steps. Silicon Graphics recommends you use swpkg to package your files for installation. See the *Software Packager User's Guide* for information for instructions on using swpkg. See "Step Seven: Updating Your Installation Process" on page 174 for guidelines.

Note: You cannot create your own device, host, or people icons. These are special icons used by the Desktop and can currently be created only by Silicon Graphics.

Creating an Icon: The Basic Steps Explained in Detail

This section describes in detail each of the basic steps listed in "Checklist for Creating an Icon" on page 163. The steps are:

- "Step One: Tagging Your Application"
- "Step Two: Drawing a Picture of Your Icon"
- "Step Three: Programming Your Icon"
- "Step Four: Compiling the Source Files"
- "Step Five: Installing Your Application in the Icon Catalog"
- "Step Six: Restarting the Desktop"
- "Step Seven: Updating Your Installation Process"

Step One: Tagging Your Application

The first step is to tag the application or shell script with its own unique identification number so that the Desktop has a way of matching the application with the corresponding FTRs. The easiest way to tag your application is to use the tag command. In order to use tag, your application must be an executable or a shell script, and you must have write and execute permissions for the file.

Note: You do not tag data or configuration files used by your application. Instead, you provide rules as described in "Matching Files Without the tag Command" on page 214 to identify these files.

If your application does meet the criteria for using the tag command, then select a tag number from your block of registered tag numbers. If you do not have a block of registered tag numbers, you can get one by calling 415/933-TAGS or sending an e-mail request to Silicon Graphics at this mail address:

```
desktoptags@sgi.com
```

After Silicon Graphics sends you a block of registered tag numbers, use the tag(1) command to assign one to your application. To do this, change to the directory containing your application and enter:

% tag tagnumber filename

where *tagnumber* is the number you assign to the application and *filename* is the name of the application.

For example:

```
% /usr/sbin/tag 0X0101011 myapp
```

For more detailed information on the tag command, see the tag(1) reference page.

Step Two: Drawing a Picture of Your Icon

The next step is to create the picture for your icon. An icon picture generally consists of a unique **badge** plus a generic component (for example, the "magic carpet" designating executables). The badge is the part of the icon picture that appears in front of the generic component and that uniquely identifies your application. The generic components are

pre-drawn and installed by default when you install the IRIX Interactive Desktop environment.

"Designing the Appearance of Icons" in Chapter 2 of the *IRIX Interactive User Interface Guidelines* provides guidelines for drawing your icon images. If possible, consult with a designer or graphics artist to produce an attractive, descriptive icon. Chapter 12, "Using IconSmith," describes exactly how to draw such an icon. Save the badge in a file called <IconName > . fti, where *IconName* is any name you choose. Choose a meaningful name (such as the name of the application or data format). If you have separate pictures representing the open and closed states of the icon, it's a good idea to name them <IconName > . open.fti and <IconName > . closed.fti, respectively.

After drawing your badge with IconSmith (described in Chapter 12) save the picture—the filename should end in .fti—and put the saved file in the correct directory. The appropriate directory depends on where you put your FTR files:

- If you put your FTR (.ftr) files in the /usr/lib/filetype/install directory (where you typically should install your FTR files), then put your badge (.fti) files in the /usr/lib/filetype/install/iconlib directory.
- If you put your FTR files in one of the other directories listed in Appendix F, then put your badge file in a subdirectory of that directory. Name the subdirectory iconlib if the subdirectory doesn't already exist.

Step Three: Programming Your Icon

Programming an icon means creating a file type. Each file type consists of a set of file typing rules, each of which defines some aspect of the look or behavior of the icon. Your file type includes rules that name the file type, and tells the Desktop where to find the associated icon files, what to do when users double-click the icon, and so on. Chapter 13, "File Typing Rules," describes how to create the FTR file that defines your file type. "Defining the Behavior of Icons With FTRs" in Chapter 2 of the *IRIX Interactive User Interface Guidelines* describes the types of behaviors your icons should support.

(This section assumes that you are writing your FTRs completely from scratch. You may prefer instead to modify an existing file type. To learn how to find the FTRs for an existing icon, see "Add the FTRs: An Alternate Method" on page 168.)

Where to Put FTR Files

Most FTR files that are not created at Silicon Graphics belong in the /usr/lib/filetype/install directory. There are also specific FTR directories set aside for site administration. For a list of all FTR directories, see Appendix F, "."

If you want to have a look at some existing FTR files, check out the /usr/lib/filetype/install directory.

Naming FTR Files

If you have an existing FTR file, you can add the new file type to this file. Otherwise, you need to create a new FTR file, which you should name according to the standard naming convention for application vendors' FTR files. The convention is:

```
vendor-name[.application-name].ftr
```

where **vendor-name** is the name of your company and **application-name** is the name of your application.

Name the File Type

Each file type must have a unique name. To help insure that your file type name is unique, base it as closely as possible on your application name.

As an extra check, you can search for your file type name in the /usr/lib/filetype directory, to make sure that the name is not already in use:

1. Change to the /usr/lib/filetype directory:

```
% cd /usr/lib/filetype
```

2. Search for the file type name:

```
% grep "your_name_here" */*.ftr
```

where *your_name_here* is the name you've selected for your file type.

If you find another file type of the name you have chosen, pick a new name.

Add the FTRs

To create a file type, either add the file type definition to an existing FTR file or create a new FTR file. You can define all the necessary file types for your application in a single FTR file.

Each file type definition **must** include the following rules:

• the TYPE rule, to tell the Desktop that you are declaring and naming a new type (the TYPE rule must go on the first line of each filetype definition)—a type is a unique type of icon, such as an email icon. For example, the file myftrs.ftr contains two filetypes:

```
TYPE FOO

MATCH ...

TYPE FOO2

MATCH ...
```

- the LEGEND rule, to provide a text description when users view icons as a list
- the MATCH rule, to allow the Desktop to match files with the corresponding file type
- the ICON rule, to tell the Desktop how to draw the icon to use for this file type

Note: The TYPE, LEGEND, and other rules are typically referred to as "rules," and the entire set of rules defining a single file type is called a "filetype," or an "FTR."

In addition to these basic components, you can add other rules as necessary for each different filetype you define.

Add the FTRs: An Alternate Method

If you don't want to write the file type from scratch, you can modify an existing file type.

The first step is to choose a file type that produces icon behavior similar to what you want from your new file type (that is, does the same thing when you double-click the icon, acts the same way when you drop the icon on another icon, and so on.)

To find the set of FTRs that define the file type for the an icon, first locate the icon on the Desktop. If the icon isn't already on the Desktop select "File QuickFind" from the Find

toolchest and use the Find an Icon window to find the icon. (When the icon appears in the drop pocket, drag it onto the Desktop.

Select the icon by clicking the left mouse button on it, then hold down the right mouse button to get the Desktop menu. When the menu appears, select the "Get Info" menu item. A window appears. In the window, look at the line labeled, "Type."

For example, if you'd selected the jot icon, the line would read:

```
Type: jot text editor
```

The string "jot text editor" is produced by the LEGEND rule. You can use this string to find the FTR that defines the jot file type. To do this, open a shell and follow these steps:

- 1. Change to the /usr/lib/filetype directory
 - % cd /usr/lib/filetype
- 2. Search for "jot text editor"

```
% grep "jot text editor" */*.ftr
```

The system responds with this line:

```
system/sgiutil.ftr: LEGEND :308:jot text editor
```

This line tells you that the jot FTR is in the /usr/lib/filetype/system directory in a file named sgiutil.ftr. The :308: is to allow the jot LEGEND to be localized (translated into languages other than English). The 308 refers to line number 308 in the uxsgidesktop message catalog. See the gettxt(1) reference page for a description of how to use such message files. If you only intend to ship your application in English speaking countries, you can omit the line number designation (for example, :308:) from your filetype.

Now you can open the sgiutil.ftr file using the text editor of your choice, and search for the "jot text editor" string again. This shows exactly where the jot FTR is in the sgiutil.ftr file.

Note: If the jot file type did not have its own icon, this search would not give you the filename.

Now you can go to the file with the jot FTR and copy it into the FTR file for your new file type. Then rename and modify the copied FTR to fit your new file type, as described in "Step Three: Programming Your Icon" on page 166.

An Example File Type

Here is an example of a simple file type:

```
TYPE ScrimShaw

MATCH tag == 0x00001005;
LEGEND the scrimshaw drawing program
SUPERTYPE Executable
CMD OPEN $LEADER
CMD ALTOPEN launch -c $LEADER
ICON {
    if (opened) {
        include("../iconlib/generic.exec.open.fti");
    } else {
        include("../iconlib/generic.exec.closed.fti");
    }
    include("/iconlib/scrimshaw.fti");
}
```

Here's a brief description of what each of these lines does:

- The first line contains the TYPE rule, which you use to name the file type. In this case, the file type is named, **ScrimShaw**. Always place the TYPE rule on the first line of your FTR. The TYPE rule is described in "Naming File Types: The TYPE Rule" on page 210.
- The second line contains the MATCH rule. Use the MATCH rule to tell the Desktop which files belong to this file type. In this example, we are just writing in the identification (tag) number that we have already assigned to the executable. The MATCH rule is described in "Matching File Types With Applications: The MATCH Rule" on page 212.
- The third line contains the LEGEND rule. Use this rule to provide a brief descriptive phrase for the file type. This phrase appears when users view a directory in list form. It also appears when users select the "Get File Info" item from the Desktop pop-up menu. In this case, the descriptive phrase is "the scrimshaw drawing program." The LEGEND rule is described in "Adding a Descriptive Phrase: The LEGEND Rule" on page 218.
- The fourth line contains the SUPERTYPE rule. Use this rule to name a file type superset for your FTR. In this example, the SUPERSET is "Executable." The SUPERTYPE rule is described in "Categorizing File Types: The SUPERTYPE Rule" on page 210.

- The fifth line contains the CMD OPEN rule. This rule tells the Desktop what to do when users double-click the icon. In this example, double-clicking the icon opens the scrimshaw application. The \$LEADER variable is a Desktop variable, usually set to the full name of the first selected icon. The Desktop variables are listed and defined in Appendix B, "." The CMD OPEN rule is described in "Programming Open Behavior: The CMD OPEN Rule" on page 219.
- The final lines contain the ICON rule. These lines tell the Desktop where to find the generic components (open and closed) as well as the unique application-specific badge. The generic components together with the badge comprise the scrimshaw icon appearance. Note that this rule combines the generic component for open and closed executables with the unique "scrimshaw" badge that identifies it as a distinctive application. The ICON rule is described in "Getting the Icon Picture: The ICON Rule" on page 227.

Step Four: Compiling the Source Files

The Desktop compiles FTR source files into files called .otr files. These files are in the /usr/lib/filetype directory.

Note: Users can create personal desktop.otr files. See /usr/lib/filetype/Makefile.personal and *Desktop User's Guide* for information about how to create and use personal desktop icons.

Note: The .ctr files are obsolete as of IRIX release 6.3.

Any time you add or change FTRs (or print conversion rules) you must recompile the .otr files by following these steps:

1. Change to the /usr/lib/filetype directory:

```
% cd /usr/lib/filetype
```

2. Become superuser:

% su

3. Recompile the files:

```
# make -u
```

(If you don't use the **-u** option when you make the files, some of your changes may not take effect.)

To activate the new FTRs, quit and restart the Desktop. For instructions on restarting the Desktop, see "Step Six: Restarting the Desktop" on page 174.

Step Five: Installing Your Application in the Icon Catalog

It's easy to add your icon to the icon catalog. Just install a symbolic link to your application in the /usr/lib/desktop/iconcatalog/C/Applications directory.

Note: The iconbookedit command is obsolete as of IRIX release 6.3.

Then, add the install rule in your Makefile. For example, enter:

```
/etc/install -idb myIdbTag -F \
/usr/lib/desktop/iconcatalog/pages/C/Applications -lns /usr/sbin/myapp MyApp
```

where C is the \$LANG environment variable, Applications is the page on which the icon will appear in the icon catalog, and MyApp is the name to appear under your icon in the icon catalog (the name can be different from the name of the executable).

Another example:

```
/etc/install -idb myIdbTag -F /usr/lib/desktop/iconcatalog/pages/C/WebTools \ -lns /usr/sbin/mywebapp MyWebApp
```

In this example, the application, mywebapp, will appear on the WebTools icon catalog page, with the name MyWebApp under the icon.

For additional information, see "Making Application Icons Accessible" in Chapter 2 of the *IRIX Interactive User Interface Guidelines*, which describes the Icon Catalog and how to select the appropriate page of the Icon Catalog for your application.

Step Six: Restarting the Desktop

To view your changes and additions, you must restart the Desktop. To restart the Desktop, first kill it by typing:

```
% /usr/lib/desktop/telldesktop quit
```

Then, restart the Desktop by selecting "In my Home Directory" from the Access Files menu on the Desktop toolchest.

Step Seven: Updating Your Installation Process

Silicon Graphics recommends you use swpkg to package your files for installation. The *Software Packager User's Guide* describes how to package your application for installation.

Your installation process must:

• Tag the executables it produces ("Step One: Tagging Your Application" on page 165 explains how to tag executables). Put the tag in the Makefile before the command that installs your application in the appropriate directory. For example:

• Copy .fti and .ftr files to the appropriate directories ("Where to Put FTR Files" on page 167 and "Where to Install Your Completed Icon" on page 178 explain which directories these files belong in). With swpkg, you can do this by setting the appropriate destination directory and destination filename for each file, using the Edit Permissions and Destinations worksheet. See Chapter 5, "Editing Permissions and Destinations," in the Software Packager User's Guide for instructions.

- Invoke make in /usr/lib/filetype to update the Desktop's database ("Step Four: Compiling the Source Files" on page 172 explains how to update the database). With swpkg, you can do this using the exitop attribute from the Add Attributes worksheet. Set up the exitop attribute to run the make command. See Chapter 6, "Adding Attributes," in the Software Packager User's Guide for instructions.
- Add your icon to the Icon Catalog, creating a symbolic link in the /usr/lib/desktop/iconcatalog/C/Applications directory. See "Step Five: Installing Your Application in the Icon Catalog" on page 173 for instructions.

See the make(1), sh(1), and tag(1) reference pages for more information on these commands.

Using IconSmith

This chapter explains how to use the IconSmith tool to draw an icon for your application. This chapter contains these sections:

- "About IconSmith" on page 178 briefly describes the IconSmith tool.
- "Where to Install Your Completed Icon" on page 178 explains where to put your icon file, after you've finished drawing your icon.
- "Some Definitions" on page 179 defines some terms you'll need to use IconSmith.
- "Starting IconSmith" on page 180 explains how to start the IconSmith tool.
- "IconSmith Menus" on page 181 discusses IconSmith's main menus: the IconSmith menu and the Preview menu.
- "IconSmith Windows" on page 182 lists IconSmith's windows: the main window, the Palette window, the Constraints window, and the Import Icon (Set Template) window.
- "Drawing With IconSmith" on page 183 describes IconSmith's drawing tools.
- "Selecting" on page 189 describes IconSmith's selection features.
- "Transformations" on page 191 describes IconSmith's transformation features.
- "Concave Polygons" on page 193 explains how to construct concave polygons in IconSmith.
- "Constraints: Gravity (Object) Snap and Grid Snap" on page 194 explains how to use IconSmith's gravity snap and grid snap features to guide your drawing.
- "Icon Design and Composition Conventions" on page 196 explains how to make sure that your icon complies with the basic icon design and composition conventions described in "Designing the Appearance of Icons" in Chapter 2 of the IRIX Interactive User Interface Guidelines.
- "Advanced IconSmith Techniques" on page 198 describes some advanced techniques, such as drawing circles and ovals in IconSmith.

About IconSmith

IconSmith is a program for drawing Desktop icons. Icons drawn with IconSmith are saved in an **icon description language**. The icon description language is described in Appendix D, "."

Designed for the specific requirements of the Desktop, IconSmith produces icons that draw quickly and display properly on the Desktop on all Silicon Graphics workstations.

An icon picture generally consists of a unique **badge** plus a generic component (for example, the "magic carpet" designating executables). The badge is the part of the icon picture that appears in front of the generic component and that uniquely identifies your application. The generic components are pre-drawn and installed by default when you install the IRIX Interactive Desktop environment.

You don't need to draw the generic components of your icons. When using IconSmith to draw your icon badge, you can import the generic component as a template as described in "Importing Generic Icon Components (Magic Carpet)" on page 196. Then your ICON rule in your FTR can include the generic components so they appear behind your badge(s) when the icon is rendered on the Desktop.

Note: Iconsmith is not a general-use drawing application. Use it only to draw Desktop icons.

Where to Install Your Completed Icon

After drawing your badge with IconSmith, save the badge—the filename should end in .fti—and install the saved file in the correct directory:

- If you install your FTR (.ftr) files in the /usr/lib/filetype/install directory (where you typically should install your FTR files), then install your icon (.fti) files in the /usr/lib/filetype/install/iconlib directory.
- If you install your FTR files in one of the other directories listed in Appendix F, then
 install your badge in a subdirectory of that directory. Name the subdirectory
 iconlib if the subdirectory doesn't already exist.

After you install the badge in the appropriate directory, see the following for more information:

- "Step Three: Programming Your Icon" on page 166
- "Step Four: Compiling the Source Files" on page 172
- "Step Five: Installing Your Application in the Icon Catalog" on page 173
- "Step Six: Restarting the Desktop" on page 174
- "Step Seven: Updating Your Installation Process" on page 174

Some Definitions

IconSmith uses some terms that may not be familiar to you. This section defines some terms used in the rest of this chapter.

Caret

The **caret** (shown in Figure 12-1) is a small red and blue cross. The caret always shows the location of the last mouse click—when you click the left mouse button, the caret appears where the cursor is pointed. Unlike the cursor, the caret shows the effects of grids and gravity (described in "Constraints: Gravity (Object) Snap and Grid Snap" on page 194).



Figure 12-1 Caret

Transformation Pin

The **transformation pin** (shown in Figure 12-2) indicates the point from which an object is scaled or sheared and around which an object is rotated. It is a blue and white cross, larger than the caret. It can be dropped anywhere to affect a transform.



Figure 12-2 Transformation Pin

Vertex

A **vertex** (shown in Figure 12-3) is a selectable point, created when the mouse is clicked in the IconSmith window while the **<Ctrl>** key is held down.



Figure 12-3 Vertex

Path

A **path** (shown in Figure 12-4) is one or more line segments between vertices. Paths can be open or closed. A closed path can be filled or unfilled.



Figure 12-4 Path

Starting IconSmith

To start IconSmith from the Desktop, double-click the IconSmith icon, shown in Figure 12-5.



Figure 12-5 The IconSmith Icon

To start IconSmith from the command line, type:

% /usr/sbin/iconsmith

IconSmith Menus

The IconSmith main window, shown in Figure 12-6, provides two popup menus, the IconSmith menu and the Preview menu:

- Access the IconSmith menu by holding down the third (typically the right) mouse button anywhere in the main window drawing area.
- Access the Preview menu by holding the third (right) mouse button down within the blue preview square located in the lower left corner of the IconSmith main window.

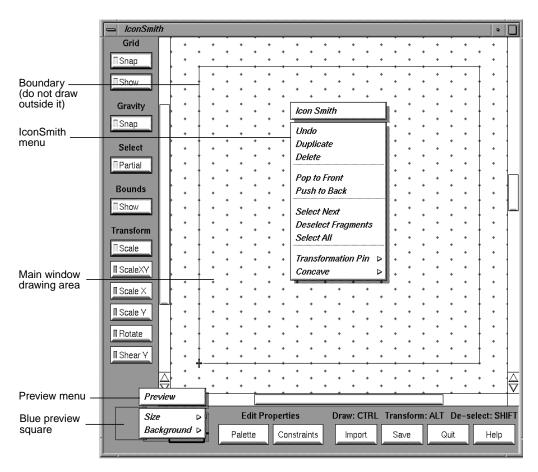


Figure 12-6 The Main IconSmith Window With Popup Menus

IconSmith Windows

Besides the main window, IconSmith provides three other primary windows:

- **Palette** (Selection Properties) window, which is described in "Drawing Filled Shapes."
- **Constraints** window, which is described in "Constraints: Gravity (Object) Snap and Grid Snap."

• Import Icon or Set Template window, which is described in "Templates."

Drawing With IconSmith

IconSmith provides tools for drawing paths, selecting colors, importing design elements from other icons, drawing shapes, and using template images. This section covers the following topics:

- "Drawing Paths"
- "Deleting"
- "Keeping the 3-D Look"
- "Drawing Filled Shapes"
- "Sharing Design Elements"
- "Templates"

Before you begin drawing, it's often useful to set up the preview box to represent the Desktop as closely as possible. This helps you choose colors and draw your icon to look its best when it appears on the Desktop and in the Icon Catalog. To do this:

- 1. Set the Background color to WorkSpace from the Preview popup menu. This gives the preview area the background color of the Icon Catalog.
- 2. Click the *Import* button at the bottom of the IconSmith window to display the Import Icon or Set Template window (shown in Figure 12-7).
- 3. Import an appropriate template from the Import menu. For example, if you are drawing an icon for a new application, import the Closed Application template by clicking that button. This helps you center and size your design appropriately.

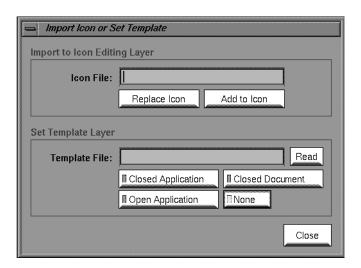


Figure 12-7 The Import Icon or Set Template Window

Tip: Draw your icon design on a clear transparency, and tape the transparency to the top of the monitor housing. Using the drawing as a guide, trace it using IconSmith.

Tip: When drawing in IconSmith, it is easy to select the wrong object. One technique that you can use is to draw adjacent icon components separately to prevent confusion when selecting and editing an object. When you have finished working with the parts, you can move them together.

There is an "Undo" option in the IconSmith popup menu. To bring up the IconSmith popup menu, hold down the right mouse button. You can undo up to nine operations using the <fl>
<fl>key. To redo something you have undone, hold the <shift> key and press the <fl>
<fl>key.

No single polygon can contain more than 255 vertices.

Also, be careful not to draw outside the royal blue boundary that appears in the preview box and in the drawing area. The Desktop doesn't display correctly outside those areas.

Drawing Paths

To draw a path with IconSmith:

- 1. Select a starting point by clicking the primary (usually the left) mouse button.
- 2. Move the mouse to a new position.
- 3. Hold down the **<Control>** key and click the primary (left) mouse button.

This process creates a line segment. To add more line segments connected to the first, repeat steps 2 and 3 as many times as necessary. To create a disconnected line segment, repeat from step 1.

Drawing Filled Shapes

In IconSmith, you can fill a closed path (one in which the beginning and end points meet) with a color. To begin, click the *Palette* button at the bottom of the IconSmith window to display the Palette window, shown in Figure 12-8.

To draw a filled shape, select a fill color from the Palette menu, and proceed to draw. When you finish creating the closed path, the shape is filled with the current fill color. To change the fill color of an existing polygon, select it by clicking on one of its vertices. Then choose a new fill color from the Palette. Also, you can change the fill color of a path by selecting the path and then selecting a new fill color.

Fill does not work properly with concave closed paths, nor with paths in which the beginning point does not meet the end point. See "Concave Polygons" on page 193.

The default fill color is "Icon," a special white color, and the default line color is "Outline," a special black color (see "Selecting Colors" on page 197).

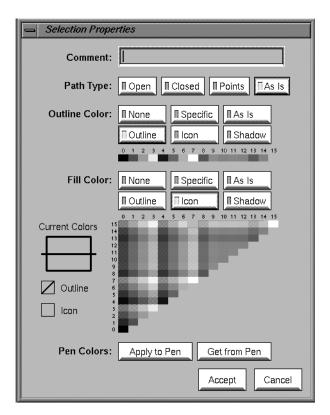


Figure 12-8 The Palette (Selection Properties) Window

Deleting

To delete any path or vertex, select it and press the <Back Space> key, or use "Delete" in the IconSmith popup menu.

Keeping the 3-D Look

Icons created by Silicon Graphics are drawn in the same **isometric view**, which provides an illusion of 3-D, even though the polygons composing the icons are 2-D. If you draw icons facing the screen at right angles, they look 2-D. To generate a 3-D effect, draw "horizontal" lines so that they move up 1 unit in the y-axis for every 2 units they extend along the true x-axis. See Figure 12-9.

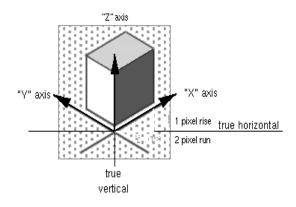


Figure 12-9 3-D Icon Axes

Use the same projection that the original icon set uses. Icons tilted in the wrong direction look off-balance, and destroy the 3-D appearance. For your convenience, IconSmith provides an isometric grid. By following the diagonals of this grid, as shown, you can create an icon that fits in exactly with other isometric icons in the Desktop. You can count along these diagonal grid dots, to help measure, align, or center pieces of your icon.

Drawing for All Scales

Desktop icons can be displayed in many sizes. IconSmith includes two features useful in designing your icon for display at all sizes, the Preview box and the slider on the right side of the drawing area.

The Preview Box

You can use the **Preview box** to see your icon design in common sizes and background colors. The Preview box is the blue box in the lower left corner of the main IconSmith window. By default, the Preview box shows your drawing at the default Desktop icon display size and no background color. You can change the icon size and background color in this window using the Preview box popup menu.

As you design your icon, periodically check its appearance in the Preview box. Because users can enlarge icons only to a maximum size of about 1x1 inch, many details will not appear or will become distorted at normal icon size. Also, keep in mind that the more detail your icon has, the longer it takes for the Desktop to render the icon.

Changing Drawing Size

You can change the size of your design in the IconSmith drawing area using the slider on the right side of the drawing area. Use the **slider** to look at your design at all sizes. At particularly small sizes, some features may not be visible. At large sizes, design imperfections may appear.

Sharing Design Elements

You can import design elements such as circles into your badge. Importing elements where possible saves you work and makes it easy to include common design elements in all the icons for one application.

To import an existing icon or icon element, click the *Import* button. This brings up the Import Icon or Set Template window. Use the "Import to Icon Editing Layer" area to specify the icon file you want.

Generic and sample material can be found in the /usr/lib/filetype/iconlib directory. For example, to import a sample circle, type the filename:

/usr/lib/filetype/iconlib/sample.circle.fti

Other icons can be found in:

- /usr/lib/filetype/default/iconlib
- /usr/lib/filetype/system/iconlib
- /usr/lib/filetype/vadmin/iconlib

All icons are potential sources for design elements. However, if you are designing a unique set of executable or document badges, you should make use of templates as described in "Templates" on page 189 and "Icon Design and Composition Conventions" on page 196.

Templates

You can use templates to help you design your icons or for tracing. You can import a template so that you can see it in the IconSmith drawing window, without saving or displaying as part of the design. This is most useful for getting position information while you are designing a unique badge to use in conjunction with the generic executable and document icons.

Note: You cannot move or change an icon template in IconSmith.

To display a template, click the **Import** button. In the Import Icon or Set Template window (shown in Figure 12-7), type the name of the template icon file you want in the textfield "Template File" in the area labeled "Set Template Layer." Alternately, click any of the three template buttons to retrieve common generic components. These template images are the most often used, and they are discussed in "Icon Design and Composition Conventions" on page 196.

Selecting

Before you edit, move, delete, or change the color of an object or vertex, you have to tell IconSmith which object you want. This section describes aspects of selecting:

- "Partial"
- "Deselect Fragments"

- "Select Next"
- "Select All"

Selecting can be difficult in a complex composition. The following tips can make the task easier:

- To select an object or vertex, move the cursor on top of the object's outline and click the left mouse button. The vertices highlight blue and white when the object is selected. To move an object, double-click, hold down the left mouse button and move with the mouse. The vertices highlight green and yellow when you can move the object.
- To move a vertex, click once on the vertex, then drag it to a new location.
- You can select more than one object or vertex by holding down the <shift> key
 during the selection process. To move the objects or vertices, move only one and the
 rest will follow.
- You can select all vertices in an area with your mouse. Hold down the left mouse button and sweep the cursor across the vertices you want. The area you select is indicated by a box. When you let go of the left mouse button, all vertices are selected.
- You can deselect a vertex by holding down the **<shift>** key and clicking the vertex.

Partial

When you use the mouse to select an area with objects in it, you can include only some vertices of some objects. When you toggle on the *Partial* button, objects partially selected are highlighted. When you toggle off the *Partial* button, only objects that fall entirely within the swept-out area are selected.

Deselect Fragments

In compositions with many objects, you can use "Deselect Fragments" to make selection easier. When selecting the objects in the drawing area, you can also select adjacent objects, then deselect what you don't want. Hold the <shift> key down and click one vertex of each object you don't want. This deselects the vertex, which makes the object partially selected. Then you can use "Deselect Fragments" from the IconSmith popup menu to deselect the entire object.

Select Next

"Select Next" allows you to select a vertex that is covered by another vertex. When two or more trajectories (lines) each have a vertex at a common location, such as two triangles with a coincident edge, the "Select Next" operator is useful for selecting a trajectory other than the top one. "Select Next" is also useful in images with tiled parts, where most vertices share a location.

Select a shared vertex by clicking its location. That vertex is highlighted in yellow and green (and the red and blue caret appears at that spot). The other vertices of the trajectory selected are highlighted in white to indicate the trajectory to which the selected vertex belongs. Now each time you choose "Select Next" from the IconSmith menu, you step through all the other vertices of all the other trajectories which have a vertex at that point.

Select All

You can select all vertices in the main IconSmith window drawing area using the "Select All" option in the IconSmith popup menu. You can select all vertices in an area by holding down the left mouse button and sweeping out a box to surround the desired area.

Transformations

The Transform buttons let you shrink, enlarge, stretch, and rotate portions of your icon design. These features can make drawing easier and more precise.

To use any Transform button, follow this procedure.

- 1. Choose the Transform option you want using any of the six transform buttons located on the left side of the IconSmith window: Scale, Scale XY, Scale X, Scale Y, Rotate, or Shear Y.
- 2. Choose a point in the main IconSmith window drawing area as a reference point for the transformation by positioning the cursor and clicking the left mouse button.
- 3. Bring up the IconSmith popup menu and select "Move to Caret" from the Transform Pin rollover menu.

- 4. To select an entire object for transformation, hold down the <alt> key and double-click the object you want to transform. Otherwise, you may select individual vertices by holding down the <alt> and <shift> keys while clicking each desired vertex. Do not release the <alt> key when you have finished selecting vertices.
- 5. While still holding down the <ale> key, position the cursor inside the object you want to transform. Press and hold down the left mouse button and move the mouse to transform the object.

For example, here is how you enlarge a circle:

- 1. Choose "Scale" from the Transform menu.
- 2. Choose a point on the perimeter of the circle.
- 3. Bring up the IconSmith popup menu and select "Move to Caret" from the "Transform Pin" rollover menu.
- 4. Hold down the <alt> key and double-click the circle. All vertices on the circle are now highlighted in green and yellow.
- 5. Continue to hold down the <ale> key. Position the cursor on a vertex of the circle. Press and continue to hold down the left mouse button while you sweep the mouse out of the circle. The circle perimeter follows the cursor, enlarging the circle.
- 6. Release the left mouse button and <alt> key when the circle is the size you want.

Scale

The Scale button changes the size of an object without changing its shape.

Scale X and Y

The buttons marked *Scale X* and *Scale Y* limit scaling transformations to either horizontal or vertical, respectively. Unlike the *Scale* button, the *Scale XY* button allows you to stretch your object both horizontally and vertically.

Rotate

Using the *Rotate* button, you can rotate a selected object around the Transform Pin.

Shear Y

The *Shear Y* transformation transforms rectangles into parallelograms with one pair of sides parallel to the y axis. The *Shear Y* button is useful for transforming art that is drawn in a face-on view to an isometric view.

Note that strictly speaking, the *Shear Y* transformation performs two transformations: shear in y and scale in x.

Concave Polygons

Figure 12-10 shows a concave polygon.

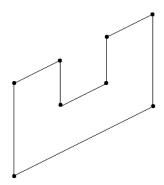


Figure 12-10 Concave Polygon

By default, IconSmith does not fill concave polygons properly. If you prefer to have concave polygons filled properly while drawing your icon design, you can tell IconSmith to draw concave polygons. Bring up the IconSmith popup menu with the right mouse button. Select "Concave" and pull out the rollover menu. Select "No GL Check" from the rollover menu. IconSmith will not check for concave polygons until you select "GL Check" from the Concave menu.

Constraints: Gravity (Object) Snap and Grid Snap

You can use **gravity snap** and **grid snap** to guide your drawing in IconSmith, allowing you to align and compose objects perfectly. This makes drawing easier and more precise. Grid snap causes the caret to "snap" to vertices or to the edges of the grid pattern displayed behind the objects you are editing. Gravity snap causes the caret to snap to vertices and the edges of objects you have already drawn. It is a good idea to make use of these features to ensure that your icon looks clean and precise at all sizes.

Typically, it's sufficient to toggle on gravity snap and grid snap. However, you can control gravity snap and grid snap properties by using the Constraints window.

Click the *Constraints* button at the bottom of the IconSmith window to display the Constraints window, shown in Figure 12-11.

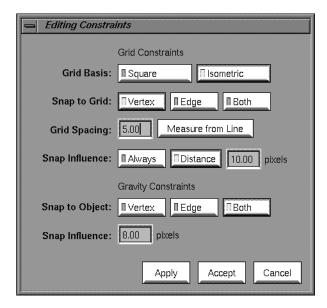


Figure 12-11 The Constraints Window

When using the Constraints window, remember to click either the *Apply* or *Accept* button to implement your changes. The *Accept* button implements your changes and closes the Constraints window, and the *Apply* button leaves the window on your screen.

Controlling the Grid

In the main IconSmith window, the *Snap* button under the heading "Grid" lets you turn on or off the grid setting you've made in the Constraints window. The *Show* button lets you display or hide the grid. To change the grid behavior, use the settings in the "Grid Constraints" portion of the Constraints window.

You can change grid properties by selecting various buttons in the Grid Constraints section of the Constraints window. Selections include:

- **Grid Basis** buttons control the shape of the grids. IconSmith includes two types of grids. The isometric grid provides guidance in the perspective described in "Keeping the 3-D Look" on page 187. IconSmith also provides a traditional square grid. To change the type of grid you are using, select a Grid Basis button, and then click the *Apply* button.
- **Snap to Grid** buttons affect what the caret gets snapped to: either vertices or edges. These changes are reflected in the appearance of the grid after you click the *Apply* button the appearance of the grid changes.
- Grid Spacing controls the distance between points in the grid. You can type in the
 number of pixels you want, or base the distance on a selected line in your icon
 design. Measure from Line measures the grid spacing from the line you select in
 the drawing area. When you copy an object using "Duplicate," the copy is placed
 one grid space down and to the right from the original (or the previous copy). You
 can use Grid Spacing to control where IconSmith places duplicate objects.
- **Snap Influence** allows you to adjust the area influenced by the "magnetic field" of the grid.

Controlling Gravity

The controls in the "Gravity Constraints" portion of the Constraints window control how gravity snap behaves. In the main IconSmith window, the *Snap* button under the "Gravity" heading lets you turn on or off the influence of gravity on objects.

- Snap to Object allows you assemble objects in your design smoothly. The object's
 edge, vertex, or both attract other objects when they are moved within range of
 gravity.
- **Snap Influence** allows you to determine the range, in pixels, of the gravity influence of objects in your design.

Icon Design and Composition Conventions

The standard set of Desktop icons has been designed to establish a clear, predictable visual language for end users. As you extend the Desktop by adding your own application-specific icons, it is important to make sure that your extensions fit the overall look of the Desktop and operate in a manner consistent with the rest of the Desktop. This section discusses:

- "Importing Generic Icon Components (Magic Carpet)"
- "Icon Size"
- "Selecting Colors"

"Designing the Appearance of Icons" in Chapter 2 of the *IRIX Interactive User Interface Guidelines* contains extensive guidelines for designing the look of your icon.

Importing Generic Icon Components (Magic Carpet)

Many icons share common components. One example is the "magic carpet" component used as a background component by most executable files; individual applications can add unique badges.

Rather than redrawing the common "generic" component in each individual icon, you can instead draw only the unique badges, and then use the ICON directive in the FTR file to combine the badge with the generic component. "Getting the Icon Picture: The ICON Rule" in Chapter 13 describes how to do this. An advantage to this approach is that you don't have to create separate icons to identify open or closed states. You can simply create the unique badge and then set up the FTR file to include either the generic open component or the generic closed component as appropriate.

While designing your icon, you can import the appropriate generic component as a template using the "Set Template Layer" of the "Import or Set Template" window; this helps you achieve the correct icon placement and perspective. When you import a component into the template layer, the template component is displayed in the drawing area, but not saved as part of the icon. When you are finished, you can save your icon in a .fti file, and combine it with the generic component in the FTR file.

If you import a generic component using the "Icon Editing Layer" section of the "Import or Set Template" window, the component becomes part of your icon. In general, you shouldn't do this; if you do, you use more disk space and icon design is more difficult.

Instead, you should draw only the badge. Then in your FTR file, you use the ICON rule to display the appropriate generic component before displaying your badge. (See "Getting the Icon Picture: The ICON Rule" on page 227 for information on the ICON rule.)

Icon Size

The blue boundary in the IconSmith drawing area indicates the area of your design that draws in the Desktop and is sensitive to mouse input. You must confine your final icon design to the area within this boundary. You can display or hide the boundary by using the *Show* button under Bounds in the main IconSmith window.

Selecting Colors

You can select or change the color of any outlined or filled object by using the features in the Selection Properties window. To bring up this window, click the *Palette* button. The currently selected outline and fill colors are displayed under the "Current Colors" heading.

There are two palettes in the Selection Properties window: one for the outline color, and another for the fill color. The outline color palette consists of the first 16 entries in the IRIS color map. The fill color palette gives you 128 colors created by dithering between the color values of the first 16 colormap entries.

In addition to the colors on these palette, there are three special colors available that you should use extensively when drawing your icon. The Desktop changes these colors to provide visual feedback when users select, locate, drag, and otherwise interact with your icon. These colors and their uses are:

Outline Color Use for outlining and line work in your icon

Shadow Color Use for contrasting drop shadows below your icon

Select outline and fill colors displayed in the palettes by clicking the appropriate buttons. If you want subsequent objects to use your color selections, click "Apply to Pen." If you want to update current objects with colors already in your pen, click an existing object with the left mouse button, and then select "Get from Pen" from the Selection Properties window. The object gets the outline and fill colors currently assigned to the pen.

On the Desktop and in the Preview box, the icon color turns yellow when the icon is selected and royal blue when an object is dropped on it. For more information on the use of color in designing icons, refer to "Icon Colors" in Chapter 2 of the IRIX Interactive User Interface Guidelines.

Advanced IconSmith Techniques

This section contains hints that make common IconSmith operations easier. This section also provides a step-by-step example of creating an icon. Topics include:

- "Drawing a Circle"
- "Drawing an Oval"
- "Isometric Circles"

Drawing a Circle

Here is a trick for drawing a circle using lines:

1. Draw a path the length of the radius of the circle you want. Figure 12-12 shows an example.

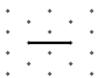


Figure 12-12 A Path

- 2. Select "Grid Spacing" of 0 pixels in the Constraints window.
- 3. Duplicate the line 12 times. Because grid spacing is set to 0, the duplicate lines stack.
- 4. Select one vertex, bring up the IconSmith popup menu, and select "Push Pin" from the Transform Pin rollover menu.
- 5. Click the *Rotate* button from the Transform menu.
- 6. Hold down the <alt> key and select the other vertex of the stack of paths.
- 7. Sweep out each path until the figure resembles a wheel, as shown in Figure 12-13.

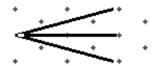


Figure 12-13 Wheel Spokes

8. Connect the outside vertices, as shown in Figure 12-14.



Figure 12-14 Connected Spokes

9. Delete the inside "spoke" paths, to get a circle like the one in Figure 12-15.

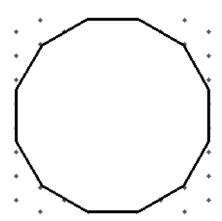


Figure 12-15 Finished 2-D Circle

Circles and other shapes can be time-consuming to create. Another way of adding circles to your icon is to import a circle from another icon or from the icon parts library. See "Sharing Design Elements" on page 188 for more information.

Drawing an Oval

To create an oval, stretch the circle you have already drawn.

- 1. Double-click a circle.
- 2. Bring up the IconSmith menu, and select "Move to Caret" from the Transform Pin menu.
- 3. Place the pin directly above the circle.
- 4. Select *Scale Y* from the Transform menu.
- 5. Hold down the <alt> key and use the mouse to stretch the circle to the oval shape you want. Figure 12-16 shows an example.



Figure 12-16 An Oval

You can now assemble the parts to make a simple icon, as shown in Figure 12-17.

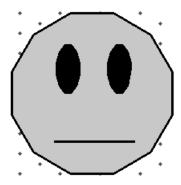


Figure 12-17 A Simple, Circular 2-D Icon

Isometric Circles

The circular icon created above is not a good central icon design because it is not isometric. The circle looks awkward in the context of isometric icons and may be misconstrued to be a sphere. Here are two ways to make the same design in isometric space.

Isometric Transformation

You can use the *Shear Y* button with an isometric grid to make any object seem 3-D.

- 1. Duplicate your circle.
- 2. Click *Shear Y* in the Transform menu.
- 3. Bring up the IconSmith menu, and select "Push Pin" from the Transform Pin menu.
- 4. Place the pin on one of the vertices at the bottom of the circle.
- 5. Hold down the <alt> key and align the bottom line of the circle using the grid.

Import Existing Object

If another icon contains the shape you need, recycle it.

- 1. Click the *Import* button.
- 2. Import the icon file /usr/lib/filetype/iconlib/sample.big.3circles.fti. You should now have the design shown in Figure 12-18 in your IconSmith drawing area.

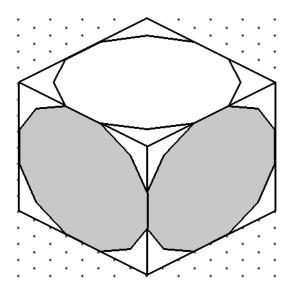


Figure 12-18 Imported Circles

3. Delete all parts of this icon except the lower right circle.

Using either method, you can create an isometric circle, shown in Figure 12-19. Starting with the isometric circle, you can easily create isometric ovals, using the procedure in "Drawing an Oval" on page 200.

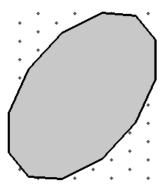


Figure 12-19 Finished Isometric Circle

The final, isometric version of the icon is shown in Figure 12-20. Note that the design still looks flat. However, if you want to show a sphere, create a straight-on circle, as was done for the WebMagic icon.

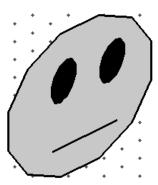


Figure 12-20 Simple, Isometric 2-D Icon

Finishing Your Icon

A finished application icon is actually three or four .fti files: one or two badges, plus generic components for the open (running) and closed (not running) icon states. You need two badges rather than one if you want to animate your icon by changing its appearance which the user double-clicks it. Figure 12-22 shows a possible open version for the example icon created in the previous section. When the icon appears on the Desktop, the generic executable icon component appears if you correctly define the ICON rule in the FTR file, as discussed in "Getting the Icon Picture: The ICON Rule" on page 227.

To see how your finished application icon appears on the Desktop:

- 1. Import the generic closed executable component using the *Import* button. In the "Import" dialogue box, under "Set Template Layer", press the *Closed Application* button. The generic icon component appears under your closed badge design.
- 2. Center your design on the generic component template you have imported, as shown in the example illustrated in Figure 12-21.

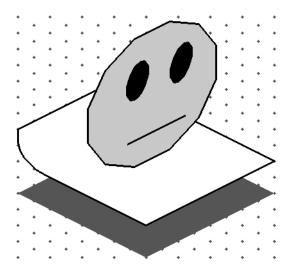


Figure 12-21 Icon Centered on Generic Component

3. (Optional, but recommended.) Follow the same two steps to create an open badge. You can give the appearance of animation by changing your design slightly and saving the changed version as an open badge.

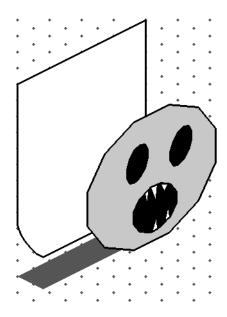


Figure 12-22 Open Icon

4. Save your icon designs to files with the suffix .fti.

For a discussion of icon file installation, see "Where to Install Your Completed Icon" on page 178. To learn how to integrate your icon into an FTR file, see "Getting the Icon Picture: The ICON Rule" on page 227.

File Typing Rules

The Desktop uses file typing rules (FTRs) to evaluate all files that are presented within the Desktop. This chapter describes each of the file typing rules in detail, and offers suggestions for good file typing style and strategies. "Defining the Behavior of Icons With FTRs" in Chapter 2 in *IRIX Interactive User Interface Guidelines* describes the behaviors your icon should support.

This chapter contains these sections:

- "A Table of the FTRs With Descriptions" on page 209 provides a reference table listing the FTRs along with brief descriptions.
- "Naming File Types: The TYPE Rule" on page 210 describes the TYPE rule, used to name a file typ.
- "Categorizing File Types: The SUPERTYPE Rule" on page 210 describes the SUPERTYPE rule, used to categorize file types.
- "Matching File Types With Applications: The MATCH Rule" on page 212 describes the MATCH rule, used to match the application with the corresponding file type.
- "Matching Non-Plain Files: The SPECIALFILE Rule" on page 217 describes the SPECIALFILE rule, used to match non-plain files.
- "Adding a Descriptive Phrase: The LEGEND Rule" on page 218 describes the LEGEND rule, used to provide a brief phrase describing the application or data file.

- "Setting FTR Variables: The SETVAR Rule" on page 219 describes how to set variables that affect the way your icon behaves.
- "Programming Open Behavior: The CMD OPEN Rule" on page 219 describes the CMD OPEN rule, used to define what happens when users open the icon.
- "Programming Alt-Open Behavior: The CMD ALTOPEN Rule" on page 220
 describes the CMD ALTOPEN rule, used to define what happens when users
 double-click your icon while pressing the <alt> key.
- "Programming Drag and Drop Behavior: The CMD DROP and DROPIF Rules" on page 222 describes the CMD DROP rule, used to define what happens when a user drags another icon and drops it on top of your application's icon.
- "Mapping Names: The MAP Rule" on page 225 describes the MAP rule, used to specify a list of all mappings from the desktop name space.
- "Programming Print Behavior: The CMD PRINT Rule" on page 226 describes the CMD PRINT rule, used to tell the Desktop what to do when a user selects your icon, then selects "Print" from the Desktop popup menu.
- "Adding Menu Items: The MENUCMD Rule" on page 226 describes the MENUCMD rule, used to add menu items to the Desktop menu
- "Getting the Icon Picture: The ICON Rule" on page 227 describes how to tell the Desktop where to find the file(s) containing the picture(s) of the icon for a file type
- "Creating a File Type: An Example" on page 230 provides a detailed example of how to program an icon.

A Table of the FTRs With Descriptions

Table 13-1 lists the rules that appear in a filetype definition along with brief descriptions.

Table 13-1 Rules That Appear in a Filetype Definition

Rules	Function		
TYPE	Declares a new type.		
SUPERTYPE	Tells the Desktop to treat the file as a subset of another type under certain circumstances.		
MATCH	Lets the Desktop determine if a file is of the declared type.		
SPECIALFILE	Tells the Desktop to use the file typing rule only on non-plain files.		
LEGEND	Provides a text description of the file type.		
SETVAR	Sets variables that affect the operation of your icon.		
CMD OPEN	Defines a series of actions that occur when a user double-clicks the mouse on an icon or selects "open" from the main menu.		
CMD ALTOPEN	Defines a series of actions that occur when a user alt-double-clicks the mouse on an icon.		
CMD DROP	Defines a series of actions that occur when a user "drops" one icon on top of another.		
DROPIF	Defines a set of file types that the icon will allow to be dropped on it.		
CMD PRINT	Defines a series of actions that occur when a user chooses "Print" from the Desktop or Directory View menus.		
MENUCMD	Defines menu entries that appear in the Desktop menu and the Selected toolchest when an icon is selected.		
ICON	Defines the appearance (geometry) of the file type's icon.		
MAP	Maps the Desktop file-type name with the ICCCM or MIME target-type names.		

All file types must begin with a TYPE rule. Aside from that, the rules can appear in any order; however, the most efficient order for parsing is to include the MATCH rule second and the ICON rule last.

Naming File Types: The TYPE Rule

It is important that your file type have a unique name so that it doesn't collide with Silicon Graphics types or types added by other developers. A good way to generate a unique file type name is to base your file type name on your application name (which is presumably unique). Another method is to include your company's initials or stock symbol in the file type name. You can also use the grep(1) command to search through existing .ftr files:

% grep name /usr/lib/filetype/*/*.ftr

Substitute your proposed new type name for the words *name*. If grep doesn't find your name, then go ahead and use it.

You name a file type by using the TYPE rule. You can define more than one file type in a single file, as long as each new file type begins with the TYPE rule. The TYPE rule always goes on the first line of the file type definition.

Here is the syntax and description for the TYPE rule:

Syntax: TYPE type-name

type-name is a one-word ASCII string. You can use a legal C language Description:

> variable as a type name. Choose a name that is in some way descriptive of the file type it represents. All rules that follow a TYPE declaration apply to that type, until the next TYPE declaration is encountered in the

FTR file. Each TYPE declaration must have a unique type name.

Example: TYPE GenericExecutable

Categorizing File Types: The SUPERTYPE Rule

Use the SUPERTYPE rule to tell other file types that your file type should be treated as a "subset" of a larger type such as executables or directories. For example, you can create an executable with a custom icon, then use the SUPERTYPE rule to tell other Desktop file types that the icon represents an executable.

Note: In general, file types don't "inherit" icons, rules, or any other behavior from SUPERTYPEs. Directories are a special case. The Desktop automatically handles the DROP, OPEN, and ALTOPEN behavior for all directories marked as "SUPERTYPE Directory." You can't override the DROP, OPEN, or ALTOPEN behavior if you include "SUPERTYPE Directory."

You can use any existing file type as a SUPERTYPE. Appendix E, "Predefined File Types", lists some of the file types defined by Silicon Graphics. You can generate a complete list of file types installed on your system using the grep(1) command:

% grep TYPE /usr/lib/filetype/*/*.ftr

Note: The list of file types generated by the above command is very long and unsorted.

Here is the syntax and description for the SUPERTYPE rule:

Syntax: SUPERTYPE **type-name** [*type-name* ...]

Description: type-name is the TYPE name of any valid file type. Use SUPERTYPE to

identify the file type as a "subset" of one or more other file types. This information can be accessed by other file types by calling <code>isSuper(1)</code> from within their CMD rules (OPEN, ALTOPEN, and so on). A file type can have multiple SUPERTYPEs. (For example, the Script file type has both Ascii and SourceFile SUPERTYPES.) See the <code>isSuper(1)</code>

reference page for more information.

Example: SUPERTYPE Executable

A common use for SUPERTYPEs is to allow users to drag data files onto other application icons to open and manipulate them. For example, if your application uses ASCII data files but you create a custom data type for those files, you can include in the file type declaration:

SUPERTYPE Ascii

This allows users to drag your application's data files onto any text editor to open and view them. If your application creates images files, you could make a similar declaration to allow users to drag data file icons to appropriate image viewers such as ipaste(1).

Matching File Types With Applications: The MATCH Rule

The Desktop needs some way to figure out which FTRs pair up with which files. Your FTRs **will not work** if they don't include some way for the Desktop to match them with the appropriate files. To do this, include the MATCH rule in your file type definition. This section explains how to use the MATCH rule to identify your files. The method you use depends on the kind of file you are matching and on the file permissions. First, here's the MATCH rule syntax and description:

Syntax: MATCH match-expression;

Description: match-expression is a logical expression that should evaluate to true if,

and only if, a file is of the type declared by TYPE. The match-expression must consist only of valid MATCH functions, as described later in this section. The match-expression can use multiple lines, but must terminate with a semicolon (;). Multiple match-expressions are not permitted for a given type. The MATCH rule is employed each time a file is encountered by the Desktop, to assign a type to that file.

Examples: MATCH tag == 0x00001005;

MATCH glob("myExecutabel");

Matching Tagged Files

The easiest way to match your application with its FTRs is to use the tag(1) command to assign a unique number to the application itself. You can then label the associated FTRs with this same unique number, using the MATCH rule, as shown in the example above.

There are a few situations in which you cannot use tag to label your files. You cannot use tag if:

- your file is neither an executable nor a shell script
- you don't have the necessary permissions to change the file

For more information on matching your files without using the **tag** command, see "Matching Files Without the tag Command" on page 214.

To tag your application and its associated FTRs using the **tag** command, follow these steps:

1. The tag command attaches an identification number to your application. Before you tag your application, select a number that is not already in use. Silicon Graphics assigns each company (or individual developer) a block of ID numbers for tagging files at no cost. If your company doesn't already have an assigned block of numbers, just send a request to Silicon Graphics. The best way is to e-mail your request to this address:

desktoptags@sgi.com

- 2. Once you have your block of numbers, you can select a number from the block of numbers assigned to your company. Make sure that you select a number that no one else in your company is using.
- 3. After you select a unique tag number for your application, go to the directory that contains your application and tag it using the tag command. This is the syntax:
 - % tag number filename

Replace the word **number** with the number that you are assigning to the application and **filename** with the name of your application. For more information on the **tag** command, see the tag(1) reference page.

4. After tagging the application itself, include the tag in your application's FTRs, using the MATCH rule. Just include a line like this in your FTR file:

```
MATCH tag == number;
```

where **number** is the unique tag number assigned to your application.

You can also use the tag command to automatically assign a tag number for a predefined file type. Silicon Graphics provides a set of generic types, called predefined types, that you can use for utilities that do not require a personalized look. These predefined file types come complete with icons, FTRs, and tag numbers. Use the appropriate tag command arguments to get the desired file type features. For more information on tag arguments, see the tag(1) reference page. The predefined file types are listed in Appendix E, "Predefined File Types".

Matching Files Without the tag Command

If you cannot use the tag command to match your application with the corresponding FTRs, you need to write a sequence of expressions that check files for distinguishing characteristics. Once you have written a sequence of expressions that adequately defines your application file, include that sequence in your FTR file, using the MATCH rule. For example, you can use this MATCH rule to match a C source file:

```
MATCH glob("*.c") && ascii;
```

The glob function returns TRUE if the filename matches the string within the quotes. The ascii function returns TRUE if the first 512 bytes of the file are all printable ASCII characters. (Table 13-3 lists all of the available match-expression functions.) The && conditional operator tells the Desktop that the functions on either side of it must both return TRUE for a valid match. See "Valid Match-Expressions" on page 215 for a list of all of the operators, constants, and numerical representations that you can use in your match-expressions.

Writing Effective Match Expressions

The most effective way to order match-expressions in a single MATCH rule is to choose a set of expressions, each of which tests for a single characteristic, and conjoin them all using "and" conditionals (&&).

The order in which you list the expressions in a MATCH rule is important. Order the expressions so that the maximum number of files are "weeded out" by the first expressions. This is advised because the conditional operator, &&, stops evaluation as soon as one side of the conditional is found to be false. Therefore, the more likely an expression is to be false, the further to the left of the MATCH rule you should place it.

For instance, in the previous MATCH expression example, it is more efficient to place the glob("*.c") expression first because there are many more ASCII text files than there are files that end in .c.

Since the Desktop scans FTR files sequentially, you must make sure that your match rule is specific enough not to "catch" any unwanted files. For example, suppose you define a type named "myDataFile" using this MATCH rule:

MATCH ascii;

Now every text file in your system will be defined as a file of type "myDataFile."

Valid Match-Expressions

This section describes the syntax and function of valid match-expressions. You can use these C language operators in a match-expression:

+	-
*	/
&	1
^	!
%	()

You can use these C language conditional operators in a match-expression:

&&	1.1
==	!=
<	>
<=	>=

The '==' operator works for string comparisons in addition to numerical comparisons.

You can use these constants in a match-expression:

true false

You can represent numbers in match-expressions in decimal, octal, or hexadecimal notation. See Table 13-2.

Table 13-2 Numerical Representations in Match-Expressions

Representation	Syntax
decimal	пит
octal	0num
hexadecimal	0xnum

Functions

Table 13-3 lists the valid match-expression functions.

 Table 13-3
 Match-Expression Functions

Function Syntax	Definition
ascii	Returns TRUE if the first 512 bytes of the file are all printable ASCII characters.
$char(\mathbf{n})$	Returns the <i>n</i> th byte in the file as a signed character; range is -128 to 127.
dircontains ("string")	Returns TRUE if the file is a directory and contains the file named by <i>string</i> . Allows use of the following expansions in <i>string</i> for pattern matching: {}[]*? and backslash (see sh(1) filename expansion). See "Using dircontains()" for more information.
glob(" string ")	Returns TRUE if the file's name matches $string$; Allows use of the following expansions in $string$ for pattern matching: { } [] * ? and backslash (see $sh(1)$ filename expansion).
linkcount	Returns the number of hard links to the file.
long(n)	Returns the n th byte in the file as a signed long integer; range is -2^{31} to 2^{31} - 1.
mode	Returns the mode bits of the file (see <i>chmod</i> (1)).
print(expr or " string ")	Prints the value of the expression <i>expr</i> or <i>string</i> to <i>stdout</i> each time the rule is evaluated; used for debugging. Always returns true.
short(n)	Returns the n th byte of the file as a signed short integer; range is -32768 to 32767.
size	Returns the size of the file in bytes.
$string(\mathbf{n},\mathbf{m})$	Returns a string from the file that is m bytes (characters) long, beginning at the n th byte of the file.
uchar (n)	Returns the n th byte of the file as an unsigned character; range is 0 to 255.
tag	Returns the specific Desktop application tag injected into an executable file by the tag injection tool (see <i>the</i> $tag(1)$ reference page.) Returns -1 if the file is not a tagged file.
ushort(n)	Returns the <i>n</i> th byte of the file as an unsigned short integer; range is 0 to 65535.

Using dircontains()

In order to use the dircontains () function, you need to include these two rules in your filetype definition:

```
SUPERTYPE SpecialFile SPECIALFILE
```

You can declare more than one SUPERTYPE in a file type, so the following would be a legal FTR file:

Predefined File Types

For some applications, you may not want to create a unique file type and icon. Several predefined file types exist and you can use them as necessary. If you use a predefined file type for your application, tag can automatically assign it a tag number. Just use the appropriate command line arguments as described in the tag(1) reference page. The predefined file types and their tag numbers are listed in Appendix E.

Matching Non-Plain Files: The SPECIALFILE Rule

SPECIALFILE is used to distinguish a file typing rule used for matching non-plain files. Device files and other non-plain files can cause damage to physical devices if they are matched using standard file typing rules (which might alter the device state by opening and reading the first block of the file).

Syntax: SPECIALFILE

Description: Special files are matched using only rules containing SPECIALFILE,

which are written so as not to interfere with actual physical devices. Similarly, plain files are not matched using rules containing a

SPECIALFILE rule.

Example: SPECIALFILE

Note: When you include the SPECIALFILE rule in your file type, you should also include the line:

SUPERTYPE SpecialFile

The SUPERTYPE declaration allows applications to use isSuper(1) to test whether your file type is a SPECIALFILE.

Adding a Descriptive Phrase: The LEGEND Rule

Use the LEGEND rule to provide the Desktop with a short phrase that describes the file type. This phrase appears when users view your icon's directory as a list. It also appears when a user selects your icon, then selects the "Get File Info" item from the Desktop menu. Make your legend simple and informative and keep it to 25 characters or less.

Here is the syntax and description for the LEGEND rule:

Syntax: LEGEND text-string

Description: text-string is a string that describes the file type in plain language that a

user can understand. Legends that are longer than 25 characters might

be truncated in some circumstances.

Example: LEGEND C program source file

You might also see a LEGEND rule that is prepended with a number between two colons—something like this:

LEGEND :290:image in RGB format

The colons and the number between them are used for internationalization. For more information, refer to "Internationalizing File Typing Rule Strings" in Chapter 4 of the *Topics in IRIX Programming*.

Setting FTR Variables: The SETVAR Rule

The SETVAR rule allows you to set variables that affect operation of your icon.

Syntax: SETVAR variable value

Description: variable is a FTR variable and value is the value to assign to the variable.

Currently, two FTR variables are supported: noLaunchEffect and noLaunchSound. Set noLaunchEffect to True to turn off the visual launch effect when the user opens your icon. Set noLaunchSound to True to turn

off the launch sound effect when the user opens your icon.

Example: SETVAR noLaunchEffect True

Programming Open Behavior: The CMD OPEN Rule

Use the CMD OPEN rule to tell the Desktop what to do when a user opens your icon. Users can open an icon in any of these ways:

- double-clicking it
- selecting it and then choosing the "Open" item from the Desktop popup menu (the
 Desktop menu is the menu that appears when you hold down the right mouse
 button while the cursor is over the Desktop background)
- selecting it and then choosing the "Open Icon" selection in the Selected tool chest.

Note: Directories are a special case. The Desktop automatically handles the OPEN behavior for all files marked as "SUPERTYPE Directory." You can't override the OPEN behavior if you include "SUPERTYPE Directory."

Here is the syntax and description for the CMD OPEN rule:

Syntax: CMD OPEN sh-expression[; sh-expression]

Description: The OPEN rule should reflect the most frequently used function that

would be applied to a file of the given type. **sh-expression** can be any valid Bourne shell expression. Any expression can use multiple lines. Any number of expressions can be used, and must be separated by semicolons (;). The final expression should not end with a semicolon. Variables can be defined and used as in a Bourne shell script, including environment variables. See Appendix B for a list of special environment variables set by the Desktop. These environment variables can be used to refer to the currently selected icons within the Desktop or Directory

View.

Examples: CMD OPEN \$WINEDITOR \$SELECTED

The CMD OPEN rule for the "Makefile" file type is more complex:

Programming Alt-Open Behavior: The CMD ALTOPEN Rule

By using the CMD ALTOPEN rule, you can tell the Desktop what to do when users double-click your icon while pressing the <alt>key.

Note: Directories are a special case. The Desktop automatically handles the ALTOPEN behavior for all files marked as "SUPERTYPE Directory." You can't override the ALTOPEN behavior if you include "SUPERTYPE Directory."

Here is the syntax and description for the CMD ALTOPEN rule:

Syntax: CMD ALTOPEN sh-expression[; sh-expression; ...; sh-expression]

Description:

The ALTOPEN rule provides added functionality for power users. Typically, you set ALTOPEN to pop up a launch window to let the user edit arguments. **sh-expression** can be any valid Bourne shell expression. Any expression can use multiple lines. Any number of expressions can be used, and must be separated by semicolons (;). The final expression should not end with a semicolon. Variables can be defined and used as in a Bourne shell script, including environment variables. See Appendix B for a list of special environment variables set by the Desktop. These environment variables can be used to refer to the currently selected icons within the Desktop or Directory View.

Examples:

```
CMD ALTOPEN launch -c $LEADER $REST
```

The CMD ALTOPEN rule for the "SGIImage" file type, defined in /usr/lib/filetype/system/sgiimage.ftr, is more complex:

```
TYPE SGIImage
CMD ALTOPEN if test -n "$IMGVIEWER"
            then
              IMGVIEWER $LEADER $REST
            else
              if test -x /usr/sbin/imgview
                then
                  imgview $LEADER $REST
                else
                    if test -x /usr/sbin/ipaste
                      ipaste $LEADER $REST
                    xconfirm -t "`gettxt uxsgidesktop:650
'/usr/sbin/imgview is missing.
Please install imgtools.sw.tools.'`" \
               -B "`gettxt uxsgidesktop:736 'Continue'`" >
/dev/null
                     fi
                 fi
          fi
```

In the previous example:

- This filetype uses the IMGVIEWER environment variable. This is one of six environment variables the user can set on the Desktop Utilities panel (/usr/sbin/dtUtilities). Your filetype should make use of such utilities as well, if appropriate. For example, if your application needs to open some data into an editor for one of its rules, then the rule should check for the WINEDITOR variable and use it to open the data. For more information on dtUtilities, see Chapter 10, "Using the Default Viewer and Editor Utilities Panel."
- This filetype posts a dialog if it can't find any appropriate applications in which to open the SGIImage file. This dialog has text that you can translate into other languages; the embedded gettxt commands retrieve the text in the appropriate language from the specified uxsgidesktop message catalog. You can make a message catalog to hold your own messages, then use gettxt to make your filetypes use these messages. For more information, see Chapter 11, "Step Three: Programming Your Icon."

Programming Drag and Drop Behavior: The CMD DROP and DROPIF Rules

Users can perform certain functions by dragging an icon and dropping it on top of another icon. 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. You use the CMD DROP rule to tell the Desktop what to do when a user drags another icon and drops it on top of your application's icon.

Note: Directories are a special case. The Desktop automatically handles the DROP behavior for all files marked as "SUPERTYPE Directory." You can't override the DROP behavior if you include "SUPERTYPE Directory."

Here is the syntax and description for the CMP DROP rule:

Syntax: CMD DROP sh-expression[; sh-expression]

Description: The DROP rule is invoked whenever a selected (file) icon is "dropped"

onto another icon in the Desktop or Directory View windows. When this happens, the Desktop checks to see if the file type being dropped upon has a DROP rule to handle the files being dropped. In this way, you can

write rules that allow one icon to process the contents of other icons. Simply drag the selected icons that you want processed and put them on top of the target icon (that is, the one with the DROP rule).

Example: CMD DROP \$TARGET \$SELECTED

By default, the CMD DROP rule handles all icons dropped on the target icon. However, if you include a DROPIF rule in your file type, only those icons whose file types are listed in the DROPIF rule are accepted as drop candidates; the Desktop doesn't allow the user to drop other types of icons on the target icon. Here is the syntax and description for the DROPIF rule:

Syntax: DROPIF file-type [file-type ... file-type]

Description: Specifies the allowable file types that a user can drop on the icon.

Example: DROPIF MailFile

Using the DROPIF rule in conjunction with the CMD DROP rule is a good practice to follow to ensure that the file types of selected icons are compatible with the selected icon. You can also use the environment variables set by the Desktop, listed in Appendix B, to determine other attributes of the selected icons.

For example, the following CMD DROP and DROPIF rules accept only a single icon with the type "MyAppDataFile":

```
DROPIF MyAppDataFile

CMD DROP if [ $ARGC -gt 1 ]
then
inform "Only one data file allowed."
else
$TARGET $SELECTED
```

In the example above, the DROPIF rule prevents users from dropping any file on the target icon except those with the type "MyAppDataFile." The CMD DROP rule is invoked only after a successful drop. It checks the value of the environment variable ARGC to see how many icons were dropped on the target icon. If more than one icon is dropped, it displays an error message; if only one is dropped, it invokes the application with the dropped file as an argument.

Note: The DROPIF rule doesn't "follow" SUPERTYPES. If you specify a file type in a DROPIF rule, only files of that type are accepted, not files that have that type as a SUPERTYPE.

If you want to handle all files with a given SUPERTYPE, you must use isSuper(1) to test for that SUPERTYPE in the CMD DROP rule. The following CMD DROP definition demonstrates this by accepting one or more files with an "Ascii" SUPERTYPE:

Mapping Names: The MAP Rule

If you want your datafile to work with the GoldenGate conversion, which is based on ICCCM target type names, you need to use the MAP rule to associate the desktop file-type name with the ICCCM target-type name. Then GoldenGate can access the ICCCM type.

Syntax: MAP namespace value

Description: The MAP rules specify a list of all mappings from desktop name space

to a non-desktop name. Desktop *namespace* is defined by the TYPE names. The *value* is mapped onto the desktop type, which you can obtain based on the value or TYPE name. You can use the MAP rule to translate from a desktop TYPE to another name space, such as ICCCM or MIME types. In the example below, the keyword SelectionTarget

indicates a ICCCM name space.

ICCCM Example:

```
TYPE AIFFSoundFile
MAP SelectionTarget AIFF_FILE
```

For more information about GoldenGate conversion, see Appendix G, "Using GoldenGate Data Conversion Services", and "Data Conversion Service" on page 105.

Programming Print Behavior: The CMD PRINT Rule

Use the CMD PRINT rule to tell the Desktop what to do when a user selects your icon, then selects "Print" from the Desktop popup menu. Here is the syntax and description for the CMD PRINT rule; see also Chapter 14, "Printing From the Desktop," for information on writing rules to convert your new file type into one of the printable types.

Syntax: CMD PRINT sh-expression[; sh-expression; ...; sh-expression]

Description: sh-expression can be any valid Bourne shell expression. Any expression

can use multiple lines. Any number of expressions can be used, and must be separated by semicolons (;). The final expression should not end with a semicolon. Variables can be defined and used as in a Bourne shell script, including environment variables. See Appendix B for a list of special environment variables set by the Desktop. These environment variables can be used to refer to the currently selected icons within the Desktop or Directory View. The recommended method of implementing the PRINT rule is to use routeprint, the Desktop's print-job routing utility, as in the example below. routeprint uses print conversion rules to automatically convert the selected files into formats accepted by the system's printers. See the routeprint(1) reference page for details on its syntax. See Chapter 14 for information on setting up print

conversion rules.

Example: CMD PRINT routeprint \$LEADER \$REST

Adding Menu Items: The MENUCMD Rule

Use the MENUCMD rule to add items to the bottom of both the Desktop menu and the Selected toolchest menu. The Desktop menu is the menu that appears when you hold down the right mouse button while the cursor is positioned on the Desktop. The Selected toolchest menu is the menu that appears when you hold down the left mouse button while the cursor is positioned over the Selected toolchest menu.

Menu items added to the Desktop menu and the Selected toolchest menu appear only when the icon is selected (highlighted in yellow) on the Desktop.

You can add as many menu items as you like by adding multiple MENUCMD rules to your file type. Any menu items added using the MENUCMD rule are added both to the Desktop menu and the Selected toolchest menu—you can't add menu items to just one of these menus.

Here is the syntax and description for the MENUCMD rule:

Syntax: MENUCMD "string" sh-expression[; sh-expression; ...; sh-expression]

Description: MENUCMD inserts the menu entry **string** into the Desktop or Directory

View menu if a single file of the appropriate type is selected, or if a group of all of the same, appropriate type is selected. If the menu entry is chosen, the actions described by the **sh-expressions** are performed on

each of the selected files.

Example: MENUCMD "Empty Dumpster" compress \$LEADER \$REST

You might also see a MENUCMD rule that is prepended with a number between two colons—something like this:

```
MENUCMD :472: "make install" winterm -H -t 'make install' \
-c make -f $LEADER install
```

The colons and the number between them are used for internationalization. For more information, refer to "Internationalizing File Typing Rule Strings" in Chapter 4 of the *Topics in IRIX Programming*.

To add more than one menu item to the Desktop popup menu, just add a MENUCMD rule for each item. For example, the "Makefile" file type includes all of the following MENUCMD rules:

Getting the Icon Picture: The ICON Rule

Use the ICON rule, described in this section, to tell the Desktop where to find the file(s) containing the picture(s) of the icon for a file type. The simplest way to do this is to

provide the full IRIX pathname. For example, if the .fti file is in the directory called /usr/lib/filetype/install/iconlib, you would simply write that pathname directly into your FTR file.

If you prefer not to use the absolute pathname in your FTR, you can use a relative pathname, as long as the icon file resides anywhere within the <code>/usr/lib/filetype</code> directory structure. To make use of relative pathnames, list the pathname relative to the directory containing the FTR file that contains the ICON rule. If you choose to do this, take care to keep path names used in FTR files synchronized with icon locations.

The Desktop sets Boolean status variables to indicate the state of an icon. You can use conditional statements that test these variables to alter the appearance of an icon based on its state. The state variables are: opened, which is True when the icon is opened; and selected, which is True when the icon is selected.

As described in "Importing Generic Icon Components (Magic Carpet)" in Chapter 12, a common technique is to draw a unique badge to identify an application and then combine that badge with a generic icon component. This works well if you also use conditional statements to change the appearance of an icon depending on its state. You can then combine the unique badge with a generic icon component appropriate to the icon's state. The example shown below demonstrates this technique.

Use the basic format from the example below to tell the Desktop where to find your icon files (the files that you created using IconSmith). Here is the syntax and description for the ICON rule:

Syntax: ICON icon-description-routine

ICON {

Description:

icon-description-routine is a routine written using the icon description language, detailed below. The routine can continue for any number of lines. The ICON rule is invoked any time a file of the specified type needs to be displayed in the Desktop or Directory View. The rule is evaluated each time the icon is painted by the application that needs it.

Examples:

```
if (opened) {
  include("../iconlib/generic.exec.open.fti");
  } else {
  include("../iconlib/generic.exec.closed.fti");
  }
  include("iconlib/ack.fti");
}
```

The example above shows you exactly how to write the standard ICON rule. The first line invokes the ICON rule. The next two lines tell the Desktop where to find the parts of the icon representing the open and closed "magic carpet" that makes up the generic executable icons. The unique badge is in a file named ack.fti.

Note: You must include your badge **after** including the generic component so that it appears over the generic components when displayed on the Desktop.

If you have two separate badges, one for the open and one for the closed state, your ICON rule is:

```
ICON {
    if (opened) {
        include("../iconlib/generic.exec.open.fti");
        include("iconlib/ack.open.fti");
    } else {
        include("../iconlib/generic.exec.closed.fti");
        include("iconlib/ack.closed.fti");
    }
}
```

Notice that this example gives the pathname of the icon files (.fti files) relative to the directory in which the FTR file is located. You can use the full pathname if you prefer. Your icon description routine would then look like this, assuming that ack.fti was placed in /usr/lib/filetype/install/iconlib:

```
ICON {
    if (opened) {
        include("/usr/lib/filetype/iconlib/genericexec.open..fti");
    else {
        include("/usr/lib/filetype/iconlib/generic.exec.close.fti");
    }
    include("/usr/lib/filetype/install/iconlib/ack.fti");
}
```

Creating a File Type: An Example

This section provides an example that demonstrates how to write a file type. In this example, assume we're writing a file type for a simple text editor called scribble and that we've decided on these behaviors for the scribble icon:

- When a user double-clicks the scribble icon, the Desktop runs the application.
- When a user drops another icon onto the scribble icon, the Desktop brings up the
 scribble application with the file represented by the dropped icon. Users can then
 use the scribble application to edit this file.

Note: We're making no provision for rejecting icons that represent files unsuitable for editing. You could enhance the **scribble** file type by including a line that tells the Desktop to notify users when they drop an icon of the wrong type onto the **scribble** icon.

(This section assumes that we're writing the file type completely from scratch. You might prefer instead to modify an existing file type. To learn how to find the FTRs for an existing icon, see "Add the FTRs: An Alternate Method" on page 168.)

Open an FTR File for scribble

For the purposes of this example, assume we're creating a new FTR file, rather than adding to an existing one. We just open a new file using any editor we choose, then type in whatever file typing rules we decide to use.

Add the Rules to the scribble FTR File

Now that we've opened a file for the FTRs, we just type in the rules we need to program the icon. The file type has to begin with the TYPE rule on the first line. The TYPE rule names the file type. This section discusses each line we use to create the file type.

Line 1: Name the File Type

Each file type has to have a unique name. Since our application is called scribble, assume that we decide to name the new file type "scribbleExecutable." By basing the file type name on the application name, we help insure a unique file type name.

Before using the name, **scribbleExecutable**, we search for it in the /usr/lib/filetype directory, to make sure that the name is not already in use:

1. Change to the /usr/lib/filetype directory:

```
% cd /usr/lib/filetype
```

2. Search for the name scribbleExecutable:

```
% grep "scribbleExecutable" */*.ftr
```

Assume that we do not find an existing file type with the name "scribbleExecutable," so that's what we name the new file type.

Now we use the TYPE rule to name the file type by typing this line into our FTR file:

```
TYPE scribbleExecutable
```

For more information on the TYPE rule, see "Naming File Types: The TYPE Rule" on page 210.

Line 2: Classify the Filetype

Next we use the SUPERTYPE rule to tell the Desktop what type of file the icon represents. Since **scribble** is an executable, we add this line to the FTRs:

```
SUPERTYPE Executable
```

For more information on the SUPERTYPE rule, see "Categorizing File Types: The SUPERTYPE Rule" on page 210.

Line 3: Match the File Type

Now we add the scribble executable's tag number to the file type definition by adding this line to the FTRs:

```
MATCH tag == 0 \times 00001001;
```

This step assumes that we've already tagged the executable itself with the tag 0X001001, as described in "Step One: Tagging Your Application" on page 165.

(Since scribble is an executable, we're able to use the tag command to tag it. If we were unable to use the tag command to assign an identification number to the application itself, we would need a slightly more complicated MATCH rule to match the application with its FTRs. For more information, see "Matching File Types With

Applications: The MATCH Rule" on page 212 and "Matching Non-Plain Files: The SPECIALFILE Rule" on page 217.)

Line 4: Provide a Descriptive Phrase

Next we use the LEGEND rule to provide a **legend** for the file type. The legend is a brief descriptive phrase that appears when users view a directory as a list or select "Get File Info" from the Desktop menu. It should be simple, informative, and 25 characters or less. To add the legend for **scribble**, add this line to the FTRs:

LEGEND scribble text editor

For more information on using the LEGEND rule, see "Adding a Descriptive Phrase: The LEGEND Rule" on page 218.

Line 5: Define Icon-Opening Behavior

We use the CMD OPEN rule to tell the Desktop what to do when users open the **scribble** icon. In this example we want the Desktop to run the scribble application when the icon is opened, so we include this line in the FTRs:

CMD OPEN \$LEADER \$REST

\$LEADER refers to the opened application, in this case **scribble**. The Desktop uses \$LEADER to open \$REST. In this case, \$REST means any other selected icons in the same window. \$LEADER and \$REST are Desktop environmental variables. These variables are listed and described in Appendix B, "Desktop Variables".

For more information on using the CMD OPEN rule, see "Programming Open Behavior: The CMD OPEN Rule" on page 219.

Line 6: Define Drag and Drop Behavior

We use the CMD DROP rule to tell the Desktop what to do when users drop another icon onto the **scribble** icon. In this example we want the Desktop to open the scribble application with the contents of the dropped file, so we include this line in the FTRs:

CMD DROP \$TARGET \$SELECTED

\$TARGET refers to the icon that the user dropped another icon on, in this case **scribble**; \$SELECTED refers to the icon that the user dropped onto the **scribble** icon. \$TARGET

and \$SELECTED are Desktop environmental variables. These variables are listed and described in Appendix B.

For more information on the CMD DROP rule, see "Programming Drag and Drop Behavior: The CMD DROP and DROPIF Rules" on page 222.

Line 7: Define Alt-Open Behavior

We use the ALTOPEN rule to tell the Desktop what to do when users open the **scribble** icon while holding down the **<Alt>** key. In this example, we want the Desktop to run the launch(1) program, so we include this line in the FTRs:

```
CMD ALTOPEN launch -c $LEADER $REST
```

Again, \$LEADER refers to the opened application, **scribble** and \$REST refers to any other selected icons in the same window. launch runs the launch program, and **-c** is a command line argument to launch.

For more information on the CMD ALTOPEN rule, see "Programming Alt-Open Behavior: The CMD ALTOPEN Rule" on page 220. See the launch(1) reference page for more information about using the launch command.

Line 8: Add the Icon Picture

We use the ICON rule to tell the Desktop where to find the picture for the **scribble** icon. Assume we have an icon picture in the file

/usr/local/lib/install/iconlib/scribble.fti. In this example, we add these lines to the FTRs:

```
ICON{
if (opened) {
    include("../iconlib/generic.open.fti");
} else {
    include("../iconlib/generic.closed.fti");
}
include("iconlib/scribble.fti");
}
```

These lines tell the Desktop how to find pictures for the **scribble** icon in the opened and closed states. The pathname of the icon (.fti) files is listed relative to the location of the FTR file containing the ICON rule. Relative pathnames work as long as the icon files are

located within the /usr/lib/filetype directory structure. Alternatively, you can use the absolute pathnames to the files:

- /usr/local/lib/iconlib/generic.open.fti
- /usr/local/lib/iconlib/generic.closed.fti
- /usr/local/lib/iconlib/scribble.fti

For more information on the ICON rule, see "Getting the Icon Picture: The ICON Rule" on page 227.

Name the scribble FTR File and Put It in the Appropriate Directory

Assume the name of our company is Shakespeare. Then according to the naming conventions in "Naming FTR Files" on page 167, we should name our FTR file Shakespeare.scribble.ftr. We put the file in the /usr/lib/filetype/install directory.

The scribble FTRs

Here is the set of FTRs that we created to define the file type called "scribbleExecutable."

```
TYPE scribbleExecutable
   SUPERTYPE Executable
   MATCH tag == 0x00001001;
   LEGEND scribble text editor
   CMD OPEN $LEADER $REST
   CMD ALTOPEN launch -c $LEADER $REST
   CMD DROP $TARGET $SELECTED
   ICON {
   if (opened) {
      include("../iconlib/generic.open.fti");
   } else {
      include("../iconlib/generic.closed.fti");
   }
   include("iconlib/scribble.fti"):
}
```

Printing From the Desktop

The desktop provides printing services so that users can print from an application. This chapter covers these topics:

- "About routeprint" on page 235 discusses the routeprint command, which converts files into printable form.
- "Converting a File for Printing" on page 236 explains how the Desktop converts a file for printing.
- "The Print Conversion Rules" on page 238 explains the print conversion rules.
- "The Current Printer" on page 240 discusses the Desktop's concept of the current, or default, printer and the Desktop environment variable \$CURRENTPRINTER.

About routeprint

To print a file, the Desktop invokes the routeprint(1) command. routeprint knows how to convert most files into printable form, even if the conversion requires several steps.

You can show routeprint how to convert your application's data files into printable format by adding one or more CONVERT rules to your application's FTR file.

This chapter explains the process routeprint uses to convert data files into a printable format, what file types routeprint already recognizes, and how to write your own print CONVERT rule to allow your application to tap into routeprint's powerful printing capabilities.

Converting a File for Printing

The Desktop already has rules for printing many types of files, such as ASCII, PostScript, and RGB image files. The easiest method for printing a file of arbitrary format is to break down the printing process into small, modular steps.

For example, instead of writing dozens of specialized rules to print reference pages directly for each kind of printer, you can instead convert reference pages to nroff format and then convert the nroff format to the format required for the current printer.

The diagram shown in Figure 14-1 illustrates the steps by which some of the supported Desktop file types are converted for printing. Each box represents one or more file types; the arrows between them indicate the steps by which the file types are converted. The values associated with the arrows represent the cost of the conversion. This concept is talked about more in "Print Costs" on page 237 later in this chapter.

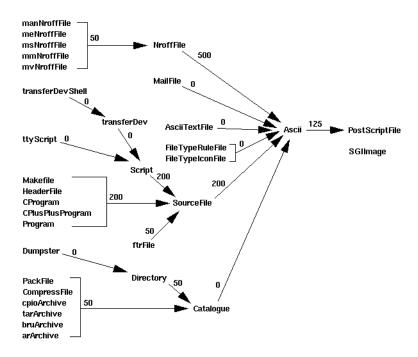


Figure 14-1 File Conversions for Printing Standard Desktop Files

This modular approach to printing has two major advantages:

- The modular steps are reusable. Because you can reuse each modular printing step, you write fewer rules.
- routeprint can pick the most efficient route for printing. There is often more than one sequence of conversion steps to print a file. routeprint chooses the sequence of steps that provides the best possible image quality.

This modular, multi-step conversion to printable form is called the **print conversion pipeline**, a series of IRIX commands that process a copy of the file's data in modular increments. The print conversion rules are designed to take advantage of this method of processing printable files.

In addition, applications or software packages can add new arcs to the CONVERT rule database whenever they define new types or have a better way of converting existing types. For example, Impressario includes a filter to go directly from NroffFile to PostScriptFile—this new filter has a lower cost than the default conversion, which goes from NroffFile to Ascii to PostScriptFile.

The Desktop already has rules for printing a large number of file types. You can use grep to list all of these print conversions definitions by typing:

```
% grep -i convert /usr/lib/filetype/*/*.ftr
```

Note: The list of print conversion definitions generated by the above command is long and unsorted.

Print Costs

Frequently, there is more than one set of steps that routeprint can use to print your file. To compare different ways of printing a file of a particular type, routeprint associates cost numbers with each conversion, then chooses the series of conversions with the lowest total cost. The cost of a conversion represents image degradation and processing cost, and is specified by a number between 0 and 1000 inclusive. The higher the cost of a conversion, the more routeprint attempts to avoid that conversion method if it has alternative methods.

The conventions for determining the cost assigned to a given conversion are described in Table 14-1.

Table 14-1 Conversion Costs for Print Conversion Rules

Cost	Reason
0	Equivalent filetypes, or a SETVAR rule (described in "The Print Conversion Rules")
50	Default conversion cost
125	Trivial data loss, or conversion is expensive
200	Minor data loss, but conversion is not expensive
300	Noticeable data loss and conversion is expensive
500	Obvious data loss (for example, color to monochrome)

The Print Conversion Rules

There are three parts to a complete print conversion rule:

- the CONVERT rule
- the COST rule
- the FILTER rule

The CONVERT Rule

Syntax: CONVERT source-type-name destination-type-name

Bescription: source-type-name is the file type you are converting from. destination-type-name is the file type you are converting to.

Example: CONVERT NroffFile PostScriptFile

Do not use the convert rule to convert directly to a new printer type; convert to a standard Desktop file type instead. Silicon Graphics reserves the right to alter printer types, so converting to a standard file type (for example, PostScriptFile) is a more portable solution. Appendix E, "," lists some of the file types defined by Silicon Graphics. You can generate a complete list of file types installed on your system using the grep(1) command:

% grep TYPE /usr/lib/filetype/*/*.ftr

Note: The list of file types generated by the above command is very long and unsorted.

The COST Rule

Syntax: COST non-negative-integer

Description: The **non-negative-integer** represents the **arc cost**, or incremental cost of

the conversion. This cost is used to reflect processing complexity or can also be used inversely to reflect the output quality. When routeprint selects a conversion sequence, it takes the arc costs into account, choosing the print conversion sequence with the least total cost. It is highly recommended that you specify a COST rule. If you omit it, routeprint assumes the cost of the conversion is zero, which may cause routeprint to return an inappropriate print conversion

pipeline. The default cost is 50.

Example: COST 50

The FILTER Rule

Syntax: FILTER filter-expression

Description: The FILTER rule represents part of an IRIX pipeline that prepares a file

for printing. **filter-expression** can be any single IRIX command line expression, and generally takes the form of one or more piped commands. In the general case, the first command within a single FILTER rule receives input from stdin; the last command in the rule sends its output to stdout. routeprint concatenates all the FILTER rules in the print conversion pipeline to form one continuous command

that sends the selected file to its destination printer.

There are three special cases in creating FILTER rules:

"first" case

"last" case

"setvar" case

In a "first" case rule, the FILTER rule is the very first rule in the print conversion pipeline. In this case, routeprint passes the list of selected files to the first command in the FILTER rule as arguments. If a first case FILTER rule begins with a command that does not accept the files in this fashion, prepend the cat command to your rule:

FILTER cat | tbl - | psroff -t

The files will then be piped to the next command's stdin.

In a "last" case rule, the FILTER rule is the very last rule in the print conversion pipeline. This rule contains a command that sends output directly to a printer (such as 1p). Last-case rules are already provided for many file types. To ensure compatibility between your application and future printing software releases, you should refrain from writing your own last-case rules. Instead, write rules that convert from your file type to any of the existing file types, and let the built-in print conversion rules do the rest.

In a "setvar" case rule, the FILTER rule is used to set an environment variable used later in the print conversion pipeline. The first CONVERT rule in the example below sets a variable that defines an nroff macro used in the second rule. In all setvar cases, stdin is passed to stdout transparently. Thus, you can include setvar as part of the pipeline in a single FILTER rule.

CONVERT mmNroffFile NroffFile COST 1 FILTER setvar MACRO=mm

CONVERT NroffFile PostScriptFile COST 50 FILTER eqn | tbl | psroff -\$MACRO -t

The Current Printer

The current printer is the system default printer that the user sets with the Print Manager or, alternatively, the printer specified by the **-p** option to routeprint. If no default is set and **-p** is not used, an error message is returned by routeprint to either stdout or a notifier window (if the **-g** option to routeprint was set). The Desktop environment variable \$CURRENTPRINTER is set to the currently selected default printer.

Example Programs for SGI Enhanced Widgets

This appendix contains example programs for some of the SGI extended IRIS IM widgets.

Makefiles are provided for some of these examples, but to use these examples, you need to:

 Link with -1Xm and -1Sgm, making sure to put the -1Sgm before -1Xm. (To replace an unenhanced widget with the enhanced version of that widget in an existing program, you need to re-link.)

```
LLDLIBS = -1Sgm -1Xm -1Xt -1X11 -1PW
```

You must include <code>-lSgm</code> to get the enhanced look and the new widgets. If you do not include <code>-lfileicon</code>, you will get a runtime error, since the runtime loader won't be able to find needed symbols. The <code>-lxm</code> represents the enhanced version of <code>libxm</code> (IRIS IM).

• Run the program with these resources:

*sgiMode: true *useSchemes: all

Set them in your .Xdefaults file or create a file for your application in /usr/lib/X11/app-defaults.

This appendix provides example programs for:

- "Example Program for Color Chooser" on page 242
- "Example Program for Dial" on page 244
- "Example Program for Drop Pocket" on page 247
- "Example Program for Finder" on page 250
- "Example Program for History Button (Dynamenu)" on page 251
- "Example Program for ThumbWheel" on page 252
- "Example Program for File Selection Box" on page 255

- "Example Programs for Scale (Percent Done Indicator) Widget" on page 258
- "Example Program for LED Widget" on page 259

Example Program for Color Chooser

```
* colortest.c --
 * demonstration of quick-and-easy use of the color
 * chooser widget.
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <Xm/Form.h>
#include <Sgm/ColorC.h>
static void ColorCallback();
Widget label, colorc;
XtAppContext app;
#if 0
int sgidladd()
 return 1;
#endif
main (argc, argv)
int argc;
char *argv[];
  Widget toplevel, form;
  Arg args[25];
  int ac = 0;
  toplevel = XtVaAppInitialize(&app, argv[0], NULL, 0, &argc, argv, NULL, NULL);
  if (toplevel == (Widget)NULL) {
   printf("AppInitialize failed!\n");
```

```
exit(1);
  colorc = SgCreateColorChooserDialog(toplevel, "colorc", NULL, 0);
  XtAddCallback(colorc, XmNapplyCallback, ColorCallback, (XtPointer)NULL);
  XtManageChild(colorc);
  form = XmCreateForm(toplevel, "Form", NULL, 0);
  XtManageChild(form);
  label = XmCreateLabel(form, "I am a color!", NULL, 0);
 XtManageChild(label);
  ac = 0;
 XtRealizeWidget(toplevel);
 XtAppMainLoop(app);
void ColorCallback(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
  Pixel white; /* fallback */
  SgColorChooserCallbackStruct *cbs =(SgColorChooserCallbackStruct *)call_data;
  Display *dpy = XtDisplay(label);
  Screen *scr = XtScreen(label);
   * If we were willing to use private structure members,
   * we could be sure to get the correct colormap by using
   * label->core.colormap. For this demo, however,
   * the default colormap will suffice in most cases.
   * /
  Colormap colormap = XDefaultColormapOfScreen(scr);
  XColor mycolor;
  Arg args[1];
  white = WhitePixelOfScreen(scr);
 mycolor.red = (unsigned short)(cbs->r<<8);</pre>
  mycolor.green = (unsigned short)(cbs->g<<8);</pre>
  mycolor.blue = (unsigned short)(cbs->b<<8);</pre>
 mycolor.flags = (DoRed | DoGreen | DoBlue);
  if (XAllocColor(dpy, colormap, &mycolor)) {
    XtSetArg(args[0], XmNbackground, mycolor.pixel);
```

```
}
else {
  fprintf(stderr, "No more colors!\n"); fflush(stderr);
  XtSetArg(args[0], XmNbackground, white);
}

XtSetValues(label, args, 1);
}
```

Makefile for colortest.c

Example Program for Dial

```
/*
 * Mytest.c --
 * create and manage a dial widget.
 * Test its resource settings through menu/button actions.
 */

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/DialogS.h>
#include <Xm/Label.h>
#include <Sgm/Dial.h>
```

```
/*
 * Test framework procedures and globals.
* /
#ifdef _NO_PROTO
static void DragCallback();
#else
static void DragCallback(Widget w, void *client_data, void *call_data);
#endif /* _NO_PROTO */
XtAppContext app;
main (argc, argv)
int argc;
char *argv[];
 Widget toplevel, form, dial, label;
 Arg args[25];
 int ac = 0;
   * Create and realize our top level window,
  * with all the menus and buttons for user input.
  toplevel = XtVaAppInitialize(&app, "Dialtest", NULL, 0, &argc, argv, NULL, NULL);
  if (toplevel == (Widget)NULL) {
   printf("AppInitialize failed!\n");
   exit(1);
 form = XmCreateForm(toplevel, "Form", NULL, 0);
  /* Set up arguments for our widget. */
 XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
  /*
  * We use all-default settings.
  * Do not set any of the dial-specific resources.
  * /
 dial = SgCreateDial(form, "dial", args, ac);
 XtManageChild(dial);
```

```
ac = 0;
XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
XtSetArg(args[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
XtSetArg(args[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
XtSetArg(args[ac], XmNtopWidget, dial); ac++;
XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_FORM); ac++;
XtSetArg(args[ac], XmNlabelString, XmStringCreateSimple("0")); ac++;
label = XmCreateLabel(form, "valueLabel", args, ac);
XtManageChild(label);
```

```
/*
   * Set up callback for the dial.
 XtAddCallback(dial, XmNdragCallback, DragCallback, label);
 XtManageChild(form);
 XtRealizeWidget(toplevel);
 XtAppMainLoop(app);
void DragCallback(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
 SgDialCallbackStruct *cbs = (SgDialCallbackStruct *) call_data;
 Widget label = (Widget)client_data;
 static char new_label[256];
 Arg args[2];
 int ac = 0;
 if ((cbs != NULL) && (label != (Widget)NULL)) {
    sprintf(new_label, "%d", cbs->position);
   XtSetArg(args[ac], XmNlabelString, XmStringCreateSimple(new_label)); ac++;
   XtSetValues(label, args, ac);
```

Example Program for Drop Pocket

```
/*
 * Demonstrate the use of the DropPocket
 */

#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Sgm/DropPocket.h>

static void droppedCB(Widget w, XtPointer clientData, XtPointer cbs ) {
    SgDropPocketCallbackStruct * dcbs = (SgDropPocketCallbackStruct *)cbs;
    char * name;
```

```
if (dcbs->iconName)
    if (!XmStringGetLtoR( dcbs->iconName, XmFONTLIST_DEFAULT_TAG, &name))
      name = NULL;
 printf("Dropped file: %s\nFull Data: %s\n", name, dcbs->iconData );
 XtFree( name );
main( int argc, char * argv[] ) {
 Widget toplevel, exitB, dp, topRC;
 XtAppContext app;
 XtSetLanguageProc(NULL, (XtLanguageProc)NULL, NULL);
 toplevel = XtVaAppInitialize( &app, "DropPocket", NULL, 0, &argc, argv, NULL, NULL);
 topRC = XtVaCreateManagedWidget( "topRC", xmFormWidgetClass, toplevel, NULL);
 dp = XtVaCreateManagedWidget("dp",
                               sgDropPocketWidgetClass, topRC,
                               XmNtopAttachment, XmATTACH_FORM,
                               XmNbottomAttachment, XmATTACH_FORM,
                               XmNleftAttachment, XmATTACH_FORM,
                               XmNrightAttachment, XmATTACH_FORM,
                               XmNheight, 100,
                               XmNwidth, 100,
                               NULL);
 XtAddCallback( dp, SgNiconUpdateCallback, droppedCB, NULL);
 XtRealizeWidget( toplevel );
 XtAppMainLoop( app );
```

Makefile for Drop Pocket Example

```
#!smake
#
include /usr/include/make/commondefs

HFILES = \\p DropPocketP.h \\p DropPocket.h

CFILES = \\p DropPocket.c
TARGETS = dpt
```

```
CVERSION = -xansi
MALLOC = /d2/stuff/lib/Malloc
CVERSION = -xansi
OPTIMIZER = -g
#-I$(MALLOC) -wlint,-pf -woff 813,826,828
LLDLIBS = -1Sqm -1Xm -1Xt -1X11 -1PW
#LLDLIBS = -u malloc -u XtRealloc -u XtMalloc -u XtCalloc -L /d2/stuff/lib
           -ldbmalloc -lSgm -lXm -lXt -lX11
LCDEFS = -DFUNCPROTO -DDEBUG
targets: $(TARGETS)
include $(COMMONRULES)
#dpt: dpTest.o $(OBJECTS)
        $(CC) -o $@ dpTest.o $(OBJECTS) $(LDFLAGS)
dpt: dpTest.o
        $(CC) -o $@ dpTest.o $(LDFLAGS)
#dpt2: dpTest2.o $(OBJECTS)
       $(CC) -o $@ dpTest2.o $(OBJECTS) $(LDFLAGS)
dpt2: dpTest2.o
        $(CC) -o $@ dpTest2.o $(LDFLAGS)
#dpt3: dpTest3.o $(OBJECTS)
        $(CC) -o $@ dpTest3.o $(OBJECTS) $(LDFLAGS)
dpt3: dpTest3.o
        $(CC) -o $@ dpTest3.o $(LDFLAGS)
#tdt: tdt.o $(OBJECTS)
        $(CC) -o $@ tdt.o $(OBJECTS) $(LDFLAGS)
tdt: tdt.o
        $(CC) -o $@ tdt.o $(LDFLAGS)
depend:
       makedepend -- $(CFLAGS) -- $(HFILES) $(CFILES)
```

Example Program for Finder

```
/*
 * Finder.c demonstrates the use of the SgFinder widget
#include <stdlib.h>
#include <stdio.h>
#include <Xm/RowColumn.h>
#include <Xm/Label.h>
#include <Sqm/Finder.h>
#include <Sgm/DynaMenu.h>
static char * items[] = { "Archer's favorite songs:",
                            "Draft dodger rag",
                            "Le Roi Renaud",
                            "/usr/sbin",
                            "/lib/libc.so.1",
                            "Calvinist Headgear Expressway",
                          };
static void valueChangeCB( Widget w, XtPointer clientData, XmAnyCallbackStruct * cbs) {
 printf("App value change callback\n");
static void activateCB( Widget w, XtPointer clientData, XmAnyCallbackStruct * cbs) {
 printf("App activate callback\n");
main( int argc, char * argv[] ) {
 Widget toplevel, rc, label, finder, history;
 XtAppContext app;
 XmString * list;
 int listSize, i;
 XtSetLanguageProc(NULL, (XtLanguageProc)NULL, NULL);
 toplevel = XtVaAppInitialize( &app, "Finder", NULL, 0, &argc, argv, NULL, NULL);
 rc = XtVaCreateWidget( "rc",
                        xmRowColumnWidgetClass, toplevel,
                        XmNresizeWidth, False,
                        XmNresizeHeight, True,
                        NULL);
  /* create the original list for the historyMenu */
 listSize = XtNumber( items );
 list = (XmString *)XtMalloc( sizeof(XmString) * listSize);
 for (i = 0; i < listSize; i++)
```

```
list[ i ] = XmStringCreateLocalized( items[ i ] );
label = XtVaCreateManagedWidget( "Things:",
                                xmLabelWidgetClass, rc,
                                NULL);
finder = XtVaCreateManagedWidget("finder", sgFinderWidgetClass, rc, NULL);
history = SqFinderGetChild( finder, SqFINDER_HISTORY_MENUBAR );
if (history && SgIsDynaMenu( history )) {
  XtVaSetValues( history,
                SgNhistoryListItems, list,
                SgNhistoryListItemCount, listSize,
                NULL);
}
for (i = 0; i < listSize; i++)
  if (list[ i ])
    XmStringFree(list[ i ]);
if (list)
  XtFree( (char *)list );
XtAddCallback( finder, XmNvalueChangedCallback, (XtCallbackProc)valueChangeCB, finder);
XtAddCallback( finder, XmNactivateCallback, (XtCallbackProc)activateCB, finder);
XtManageChild( rc );
XtRealizeWidget( toplevel );
XtAppMainLoop( app );
```

Example Program for History Button (Dynamenu)

```
main( int argc, char * argv[] ) {
 XtAppContext app = NULL;
 Widget toplevel, rc, dynaMenu;
 XmString * list;
 int listSize, i;
  toplevel = XtVaAppInitialize( &app, "DynaMenu", NULL, 0, &argc,argv, NULL, NULL);
 rc = XtVaCreateManagedWidget( "rc", xmRowColumnWidgetClass, toplevel, NULL);
  /* create the original list for the dynaMenu */
 listSize = XtNumber( items );
 list = (XmString *)XtMalloc( sizeof(XmString) * (unsigned int)listSize);
  for (i = 0; i < listSize; i++)
    list[ i ] = XmStringCreateLocalized( items[ i ] );
 dynaMenu = XtVaCreateManagedWidget("dynaMenu",
                                     sgDynaMenuWidgetClass, rc,
                                     SgNhistoryListItems, list,
                                     SqNhistoryListItemCount, listSize,
                                     NULL);
 XtAddCallback( dynaMenu, SgNdynaPushCallback, dynaPushCB, NULL);
 for (i = 0; i < listSize; i++)
    XmStringFree( list[ i ] );
 XtFree( (char *)list );
 XtRealizeWidget( toplevel );
 XtAppMainLoop( app );
```

Example Program for ThumbWheel

```
/*
 * Thumbwheel.c --
 * create and manage a thumbwheel.
 */

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/DialogS.h>
#include <Xm/Label.h>
#include <Sgm/ThumbWheel.h>
```

```
* Test framework procedures and globals.
#ifdef _NO_PROTO
static void DragCallback();
#else
static void DragCallback(Widget w, void *client_data, void *call_data);
#endif /* _NO_PROTO */
XtAppContext app;
main (argc, argv)
int argc;
char *argv[];
 Widget toplevel, form, thumbwheel, label;
 Arg args[25];
 int ac = 0;
   * Create and realize our top level window,
  * with all the menus and buttons for user input.
  * /
  toplevel = XtVaAppInitialize( &app, "Thumbwheeltest", NULL, 0, &argc, argv, NULL, NULL);
  if (toplevel == (Widget)NULL) {
   printf("AppInitialize failed!\n");
   exit(1);
 form = XmCreateForm(toplevel, "Form", NULL, 0);
  /* Set up arguments for our widget. */
 ac = 0;
 XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNtopAttachment, XmATTACH_FORM); ac++;
```

```
/*
   * We use all-default settings, with the exception of orientation.
   * Do not set any other thumbwheel-specific resources.
  ac = 0;
 XtSetArg(args[ac], XmNorientation, XmHORIZONTAL); ac++;
  thumbwheel = SqCreateThumbWheel(form, "thumbwheel", args, ac);
 XtManageChild(thumbwheel);
 ac = 0;
 XtSetArg(args[ac], XmNleftAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNrightAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNtopAttachment, XmATTACH_WIDGET); ac++;
 XtSetArg(args[ac], XmNtopWidget, thumbwheel); ac++;
 XtSetArg(args[ac], XmNbottomAttachment, XmATTACH_FORM); ac++;
 XtSetArg(args[ac], XmNlabelString, XmStringCreateSimple("0")); ac++;
  label = XmCreateLabel(form, "valueLabel", args, ac);
 XtManageChild(label);
  /*
   * Set up callback for the thumbwheel.
 XtAddCallback(thumbwheel, XmNdragCallback, DragCallback, label);
 XtManageChild(form);
 XtRealizeWidget(toplevel);
 XtAppMainLoop(app);
void DragCallback(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
 SqThumbWheelCallbackStruct *cbs = (SqThumbWheelCallbackStruct *) call_data;
 Widget label = (Widget)client_data;
 static char new_label[256];
 Arg args[2];
 int ac = 0;
 if ((cbs != NULL) && (label != (Widget)NULL)) {
    sprintf(new_label, "%d", cbs->value);
    XtSetArg(args[ac], XmNlabelString, XmStringCreateSimple(new_label)); ac++;
    XtSetValues(label, args, ac);
```

Example Program for File Selection Box

```
To run this program, add these lines to your .Xdefaults file:
                     fsb*sgiMode: true
                     fsb*useSchemes: all
                     then type:
                     xrdb -load
                     Here's the sample program:
/*----*/
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <stdlib.h>
#include <stdio.h>
#include <Xm/FileSB.h>
void printDirF( Widget w, XtPointer clientData, XmFileSelectionBoxCallbackStruct * cbs) {
  char * filename = NULL, * dirname = NULL;
 XmStringGetLtoR( cbs->value, XmFONTLIST_DEFAULT_TAG, &filename);
 XmStringGetLtoR( cbs->dir, XmFONTLIST_DEFAULT_TAG, &dirname);
 printf("Filename selected: %s\n", filename);
 if (filename)
   XtFree( filename );
 if (dirname)
   XtFree( dirname );
static void showDialog( Widget w, XtPointer clientData, XtPointer callData) {
 Widget dialog = (Widget) clientData;
 XtManageChild( dialog );
```

```
main (int argc, char *argv[]) {
 Widget toplevel, fsb, b1, b2, rc;
 XtAppContext app;
 XmString textStr;
 XtSetLanguageProc( NULL, (XtLanguageProc)NULL, NULL);
 toplevel = XtVaAppInitialize( &app, "Fsb", NULL, 0, &argc, argv, NULL, NULL);
 rc = XtVaCreateManagedWidget( "rc", xmFormWidgetClass, toplevel, NULL);
  /* Set up a dialog */
 if (argc > 1) {
   b1 = XtVaCreateManagedWidget( "FSB",
                                  xmPushButtonWidgetClass,
                                  rc,
                                  XmNtopAttachment,
                                  XmATTACH_FORM,
                                  XmNbottomAttachment,
                                  XmATTACH_FORM,
                                  XmNleftAttachment,
                                  XmATTACH_FORM,
                                  XmNrightAttachment,
                                  XmATTACH_FORM,
                                  NULL);
    fsb = XmCreateFileSelectionDialog( b1, "FSB Dialog", NULL, 0);
    XtAddCallback( b1, XmNactivateCallback, showDialog, fsb);
  } else {
    fsb = XmCreateFileSelectionBox( rc, "Select A File", NULL, 0);
    XtVaSetValues(fsb,
                   XmNtopAttachment, XmATTACH_FORM,
                   XmNbottomAttachment, XmATTACH_FORM,
                   XmNleftAttachment, XmATTACH_FORM,
                   XmNrightAttachment, XmATTACH_FORM,
                   NULL);
    XtManageChild( fsb );
```

```
XtAddCallback( fsb, XmNokCallback, (XtCallbackProc)printDirF, fsb);
XtAddCallback( fsb, XmNcancelCallback, (XtCallbackProc)exit, NULL);
XtRealizeWidget( toplevel );
XtAppMainLoop( app );
}
```

Makefile for File Selection Box Example Program

```
#!smake
#
include /usr/include/make/commondefs

CFILES = fsb.c

TARGETS = fsb

CVERSION = -xansi
OPTIMIZER = -g

LLDLIBS = -lSgm -lXm -lXt -lX11 -lPW

LCDEFS = -DFUNCPROTO -DDEBUG

LCINCS = -I. -I$(MOTIF_HEADERS)

targets: $(TARGETS)

include $(COMMONRULES)

fsb: $(OBJECTS)
$(CC) -o $@$(OBJECTS) $(LDFLAGS)
```

Example Programs for Scale (Percent Done Indicator) Widget

The following code produces a simple motif scale widget:

```
/* progress.c */
/* cc -o progress progress.c -lXm -lXt */
#include <Xm/Scale.h>
void main(int argc, char** argv) {
    Widget toplevel, scale;
    XtAppContext app_context;
   Arg args[5];
    int nargs=0;
    toplevel = XtAppInitialize(&app_context, "Progress",
                        NULL, 0, &argc, argv, NULL, NULL, 0);
    XtSetArg(args[nargs], XmNvalue, 50); nargs++;
    XtSetArg(args[nargs], XmNorientation, XmHORIZONTAL); nargs++;
    scale = XmCreateScale(toplevel, "scale", args, nargs);
    XtManageChild(scale);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app_context);
}
```

The following resource file (named "Progress") produces the slanted, thermometer look of the SGI percent done indicator. Also see the IRIS Viewkit VkProgressDialog class.

```
!Progress - App-default resources for the progress sample program
*sgiMode: true
*useSchemes: all
!Change the appearance o the slider
*scale.sliderVisual: flat_foreground
*scale.slidingMode: thermometer
*scale.slanted: true
!Set the correct scheme colors
Progress*scale*foreground: SGI_DYNAMIC BasicBackground
Progress*scale*troughColor: SGI_DYNAMIC TextFieldBackground
```

Example Program for LED Widget

```
/* ledbutton.c
/* cc -o ledbutton ledbutton.c -lXm -lXt
/* ledbutton -xrm "*sgiMode: true" -xrm "*useSchemes: all"
#include <Xm/ToggleB.h>
void main(int argc, char** argv)
  Widget toplevel, toggle;
  XtAppContext app_context;
  Arg args[5];
  int nargs=0;
   toplevel = XtAppInitialize(&app_context, "LEDButton",
              NULL, 0, &argc, argv, NULL, NULL, 0);
  XtSetArg(args[nargs], XmNindicatorSize, 10); nargs++;
  XtSetArg(args[nargs], XmNindicatorType, Xm3D_N_OF_MANY); nargs++;
  toggle = XmCreateToggleButton(toplevel, "toggle", args, nargs);
  XtManageChild(toggle);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
```

Desktop Variables

Variables that are used by the Desktop are listed below. Some of these variables can be customized by the user. You can use any of these variables as part of the OPEN, ALTOPEN, or PRINT file typing rules, or as part of the FILTER print conversion rule. In IRIX 6.3 and above, variable substitution is done by the libraries; typically you do not set environment variables. However, if a variable would expand to null, the desktop automatically sets an empty environment variable.

Variables Set By the Desktop

The variables listed below are preset by the Desktop (that is, the Desktop sets their values).

\$LEADER If one or more icons are currently selected from the Desktop, *LEADER* is

set to the icon that is selected first. If no icon is selected, it is set to null.

\$REST If more than one icon is currently selected from the Desktop, *REST*

contains the list of names of all selected icons except the highlighted icon

(see *LEADER* above). Otherwise, it is set to null.

\$LEADERTYPE

If one or more icons are currently selected from the Desktop, *LEADERTYPE* is set to the *TYPE* of the icon whose text field is

highlighted. If no icons are selected, it is set to null.

\$RESTTYPE When more than one icon is currently selected from the Desktop,

RESTTYPE contains the TYPE for all selected icons **except** the

highlighted icon, if the remainder of the selected icons are all of the same *TYPE*. If they are not the same *TYPE*, or only one icon is selected,

RESTTYPE is set to null.

\$RESTTYPELIST

Contains the list of *TYPEs* corresponding to the arguments in *REST*. If

only one icon is selected, RESTTYPELIST is set to null.

\$ARGC Contains the number of selected icons.

\$TARGET Set only for the CMD DROP rule, *TARGET* contains the name of the icon

being dropped upon; otherwise it is set to null.

\$TARGETTYPE

Set only for the CMD DROP rule, TARGETTYPE contains the TYPE of

the icon being dropped upon; otherwise it is set to null.

\$SELECTED Contains the names of the selected icons (whether or not a drop occurs

on TARGET).

\$SELECTEDTYPE

If all of the icons named in *SELECTED* are of the same *TYPE*, *SELECTEDTYPE* contains that *TYPE*; otherwise it is set to null.

\$SELECTEDTYPELIST

Contains a list of *TYPE*s corresponding to the *TYPE*s of the selected icons named in *SELECTED*. If only one icon is selected, it is set to null.

\$WINTERM

Contains the name of the window terminal invoked from the Desktop using winterm(1). Currently supported window terminals are wsh (the default) and xterm.

Variables Set By the User

The variables listed below can be set by the user and the Desktop passes these customizable variables to the FTRs. Users set these variables from the Toolchest->Desktop->Customize->Utilities panel (or by the command, dtUtilities). See "Using the Default Viewer and Editor Utilities Panel" for more information.

\$WINEDITOR Contains the path to and name for the text editor invoked from the Desktop. The default editor is /usr/sbin/jot.

\$WEBBROWSER

Contains the path to and name of an X Window System application that is a World Wide Web visual browser (for example,

/usr/bin/X11/netscape).

\$IMGVIEWER Contains the path to and name of an application that displays image files

(for example, /usr/bin/X11/imgview).

\$MAILBOXPROG

Contains the path to and name of a mail reader application (for example, /usr/bin/X11/MediaMail -gui).

\$BOOKVIEWER

Contains the path to and name of an application that displays InSight books (for example, /usr/sbin/insight).

\$PSVIEWER

Contains the path to and name of an application for viewing PostScript files (for example, /usr/bin/X11/showps).

Online Help Examples

This appendix contains listings of several online help document files.

- "A Simple Help Document"
- "Allowable Elements in a Help Document"
- "An Example of Implementing Help in an Application"

This appendix also lists the source of an example program that implements many online help features, along with its accompanying help document and helpmap file. All of these files are available online. Their locations are given before each listing.

To view these examples on your system, you must install the **insight_dev** product, which contains the SGIHelp library and include file, help generation tools, examples, and templates.

A Simple Help Document

Example C-1 lists a simple help document. It's intended as a primer for writing online help documents. You can find this file online at

/usr/share/Insight/XHELP/samples/sampleDoc/sample.sgm.

Example C-1 An Example of a Help Source File

<dochelp>

<!--

This block denotes a SGML-style comment.

For those that are unfamiliar with SGML, this sample file will try to cover the usage of a variety of the tags that are used in the XHELP DTD. The examples shown in this sample should be sufficient for a writer to produce a very high-quality, functional help document for use with an application.

It is best to view this sample once it has been published, and then compare what you see in the viewing software to the actual tags displayed in this file.

Each HelpTopic block written below displays how to use the DTD to implement specific elements/constructs. It should be fairly self-explanatory.

A couple of things to look for when constructing/editing your SGML file:

- o Make sure a starting element tag has an associated end tag! If not, then the file will not compile properly. This is analagous to missing a bracket or paranthesis in a C program!
- o SGML is NOT case sensitive! "HELPTOPIC" is the same as "helptopic", which is the same as "HelpTopic", etc.

-->

<HelpTopic HelpID="intro">
<Helplabel>SGI Sample SGML File</Helplabel>
<Description>

<para>This file contains examples using many of the constructs used
in the XHELP DTD.</para>

<para>Notice that the general outline used for putting together
a help "card" is defined by this particular SGML block. The preceding tag
defines the title that will be displayed for this card. The area you
are currently reading is a description for the feature or function you
are documenting. It is not necessary to use each of these tags, although

```
the "HelpTopic" tag is required.</para>
<para>A writer of help information may also wish to include a glossary
of terms. In that way, the documenter can tag terms within the text,
and have them display a specified definition from within the viewer.
A sample of this is: <glossterm>sgihelp</glossterm>.</para>
<para>The actual definition for the term is found at the end of this
SGML sample.</para>
</Description>
</HelpTopic>
<!--
 ______
 It's important to point out that the "HelpID" is the glue that
 binds the help text to the application, through the use of the
 provided Help API (library, header file).
 ______
<HelpTopic HelpID="helpid_info">
<Helplabel>What is a HelpID?</Helplabel>
<Description>
<para>The HelpID attribute is used to by your application to
instruct the help server which help "card" to display. In this
case, sending the help server an ID of "helpid_info" would bring up
this particular block (or "card").</para>
<para>The other "ID" is often used as an anchor point
(and should be used within an "ANCHOR" tag) for hypertext
links within your text. If you wish to refer to a particular card
one simply uses the ID as the anchor point for the link syntax.</para>
</Description>
</HelpTopic>
<!--
 ______
 This section illustrates the simple usage of specifying a note,
 warning, tip, or caution within your help document.
 ______
<HelpTopic HelpID="note_example">
```

```
<Helplabel><Anchor Id="AI003">Using Notes, Warnings or Tips Within a Paragraph/Helplabel>
<Description>
<para>Within the paragraph tag, there are a variety of text marking
mechanisms. Each of these delineations must appear as part of the
paragraph ("para") element.
<para>This area shows the documentor how a warning, note or "tip"
can be used within a persons's help text.</para>
<para>
<warning><para>Be Careful. This is a warning.</para></warning>
<note><para>For your information, this is a note.</para></note>
<tip><para>When you prepare your help file, you may wish to include a tip.</para>
</tip>
<caution><para>Use a caution tag when you wish to have the user use caution!</para>
</caution>
</para>
</Description>
</HelpTopic>
<!--
 ______
 This next section illustrates how to display computer output,
 program listings, etc. within your help document.
 ______
<HelpTopic HelpID="literal_example">
<Helplabel>Using Literals or Examples Within a Paragraph</Helplabel>
<Description>
<para>
This area shows the documentor how to implement specific examples within
their help text. It also describes how to the "literal" tag.</para>
<para>
When used within a paragraph, the LiteralLayout tag
tells the viewing software to take this next block "as is",
with all accompanying new-lines and spacing left intact.</para>
<Example>
<Title>Various Examples: ComputerOutput, LiteralLayout, ProgramListing</Title>
<para>
What follows is a computer output listing from when a
user typed <userInput>ls</userInput> :
<ComputerOutput>
```

```
% ls -1
total 6777
-rwxr-xr-x 1 guest guest
                             29452 Mar 8 19:12 menu*
-rw-r--r-- 1 guest guest
                           2375 Mar 8 19:11 menu.c++
</ComputerOutput>
</para>
<para>
Each of the subsequent three entries should be indented and on their
own line:
<LiteralLayout>
   Here is line one.
    This is line two.
   This is line three.
</LiteralLayout>
</para>
<para>
The following is a listing from a "C" program:
<ProgramListing>
    #include "X11/Xlib.h"
    #include "helpapi/HelpBroker.h"
    void main(int, char**)
         /* default to the value of the DISPLAY env var */
        Display *display = XOpenDisplay(NULL);
         if( display ) {
               /* initialize the help server */
               SGIHelpInit(display, "MyApp", ".");
         }
         . . .
</ProgramListing>
</para>
</Example>
</Description>
</HelpTopic>
<!--
```

```
______
 This next section illustrates how to incorporate graphics within
 your help text.
 ______
<HelpTopic HelpID="graphic_example">
<Helplabel>Using Graphics or Figures Within Your Help Text</Helplabel>
<Description>
<para>
This area displays how a graphics or figure can be used within the flow of
your information. The following figure is in the "GIF" format:
</para>
<Figure ID="figure_01" Float="Yes">
   <title>A GIF Raster Image</title>
   <Graphic fileref="sample1.gif" format="GIF"></Graphic>
</Figure>
<para>
Currently, support is provided for <emphasis>raster</emphasis> graphics in
the GIF and TIF formats. Support is provided for <emphasis>vector</emphasis>
graphics utilizing the CGM format.
</para>
<para>
This next figure in the CGM (Computer Graphics Metafile) format:
</para>
<Figure ID="figure_02">
   <title>A CGM Vector Image</title>
   <Graphic fileref="sample2.cgm" format="CGM"></Graphic>
</Figure>
<para>
A special note that all equations are treated as inline images, as shown
here:
<equation>
   <Graphic fileref="matrix.gif" format="GIF"></Graphic>
</equation>
</para>
</Description>
</HelpTopic>
```

```
<!--
 ______
 Hyperlinks can be a very powerful navigation mechanism!
 Liberal usage is encouraged.
 ______
-->
<HelpTopic HelpID="link_example">
<Helplabel>Using HyperLinks</Helplabel>
<Description>
<para>One of the most powerful capabilities of the sgihelp viewer
is the use of hyperlinks to associate like pieces of information.
Constructing these links in SGML is trivial.</para>
<para>Notice that the "Link" element requires an attribute called
"Linkend". This defines the area (anchor) to link to. The "Linkend"
attribute points to the ID of some SGML element. In composing
help text, it is probably best to assign an ID to each "HelpTopic"
element, and use those same ID's when specifying a Link.</para>
<para>A link is defined below:
<para>For more information about using Notes, refer to the area
entitled <Link Linkend="AI003">"Using Notes, Warnings or Tips
Within a Paragraph"</Link></para>
<para>Note that the "Anchor" tag can also be used within a
document to point to any level of granularity the author
wishes to link to.</para>
</Description>
</HelpTopic>
<!--
 Note that there are *many* ways to specify lists. This example
 shows some commonly-used permutations.
 ______
-->
<HelpTopic HelpID="list_example">
<Helplabel>Using Lists Within Your Help Text</Helplabel>
<Description>
<para>This area displays how a person can author
various types of lists within their help text.</para>
<para>Here is an itemized list that uses a dash to preface each item:
```

```
<ItemizedList Mark="dash">
<ListItem><para>First Entry</para></ListItem>
<ListItem><para>Second Entry</para></ListItem>
<ListItem><para>Third Entry</para></ListItem>
</ItemizedList>
<para>Here is an itemized list that uses a bullet to preface each item:
<ItemizedList Mark="bullet">
<ListItem><para>First Entry</para></ListItem>
<ListItem><para>Second Entry</para></ListItem>
</ItemizedList>
<para>Here is an ordered list, using standard enumeration:
<OrderedList>
<ListItem><para>First Entry</para></ListItem>
<ListItem><para>Second Entry</para></ListItem>
<ListItem><para>Third Entry</para></ListItem>
</OrderedList>
<para>Here is another ordered list, using upper-case Roman enumeration,
showing nesting (sub-items) within the list (outline format):</para>
<OrderedList Numeration="Upperroman">
<ListItem><para>First Entry</para></ListItem>
<ListItem><para>Second Entry
    <OrderedList Numeration="Upperalpha" InheritNum="Inherit">
        <ListItem><para>First SubItem</para></ListItem>
        <ListItem><para>Second SubItem</para></ListItem>
        <ListItem><para>Third SubItem</para></ListItem>
        <ListItem><para>Fourth SubItem</para></ListItem>
    </OrderedList>
</para></ListItem>
<ListItem><para>Third Entry</para></ListItem>
</OrderedList>
<para>Here is a variable list of terms:
<VariableList>
<VarListEntry>
<term>SGI</term>
<ListItem><para>Silicon Graphics, Inc.</para></ListItem>
</VarListEntry>
<VarListEntry>
<term>SGML</term>
<ListItem><para>A Meta-language for defining documents.</para></ListItem>
</VarListEntry>
</VariableList>
```

```
</Description>
</HelpTopic>
<!--
 ______
 Some final examples...
 ______
<HelpTopic HelpID="misc_example">
<Helplabel>Other Miscellaneous Textual Attributes/Helplabel>
<Description>
<para>This area displays some miscellaneous tags that can be used
within the context of your help document.</para>
<para>
<Comment>This is a comment that is not to be confused
with the SGML-style comment! Instead, this comment will be
parsed and carried into the text of your document. Usually it's
used in production, for specifying to someone an area of concern,
an area that needs editing, etc.
</Comment>
</para>
<para>Within your text, you may wish to denote a footnote.
<Footnote id="foot1"><para>This block is a footnote!</para></footnote>
The XHELP DTD will allow you to do that.
</para>
<para>
You may wish to add a copyright symbol to your text, such as:
Silicon Graphics, Inc.<trademark Class="Copyright"></trademark>
</para>
</Description>
</HelpTopic>
<!--
 ______
 If you wish to use/have a glossary of terms within your help text,
 it is advised to put it at the end of your help "book", as shown
```

```
here. NOTE: CR or other characters (#PCDATA) is NOT allowed
 between the <Glossary> and <Title> tags! (mixed content model)
 ______
-->
<Glossary>
<Title>Glossary</Title>
<GlossEntry>
<GlossTerm>help</GlossTerm>
<GlossDef>
<para>To give assistance to; to get (oneself) out of a difficulty;
a source of aid.</para>
</GlossDef>
</GlossEntry>
<GlossEntry>
<GlossTerm>sqihelp</GlossTerm>
<GlossDef>
<para>This is Silicon Graphics, Inc. version of a "Xhelp" compatible
server. Through the use of an available API, and a help text
compiler, books can be constructed that can be used to render
help information for the given application.</para>
</GlossDef>
</GlossEntry></Glossary>
<!--
 ______
 Don't forget the very last ending tag...!!!
 ______
</dochelp>
```

Allowable Elements in a Help Document

Example C-2 lists a help document that describes the legal structures defined by the help DTD. You can find this file online at

/usr/share/Insight/XHELP/samples/XHELP_elements/XHELP_elements.s gm.

Example C-2 A Description of the Elements Defined by the Help DTD

<DOCHELP>

```
<HELPTOPIC HelpID="">
<HELPLABEL>The Elements Alphabetized/HELPLABEL>
<DESCRIPTION>
<PARA>Emphasized entries indicate block-oriented elements.</PARA>
</DESCRIPTION></HELPTOPIC>
<HELPTOPIC HelpID="">
<HELPLABEL>Common Attributes </HELPLABEL>
<DESCRIPTION>
<PARA>Common attributes include ID.</PARA>
<PARA>ID is an identifier, which must be a
string that is unique at least within the document and
which must begin with a letter.</PARA>
</DESCRIPTION></HELPTOPIC>
<HELPTOPIC HelpID="">
<HELPLABEL>Other Attributes/HELPLABEL>
<DESCRIPTION>
<PARA>Certain other attributes occur regularly. PageNum is
the number of the page on which a given element begins
or occurs in a printed book. Label holds some text
associated with its element that is to be output when
the document is rendered.
Type is used with links,
as it is clear that different types of links may be
required; it duplicates the function of Role.</PARA>
<PARA>The Class attribute has been introduced in an attempt to
control the number of computer-specific in-line elements.
The elements that bear the Class attribute, such as
Interface, have general
meanings that can be made more specific
by providing a value for Class from the delimited list
for that element. For example, for the Interface element
one may specify Menu, or Button; for the MediaLabel
element one may specify CDRom or Tape. Each element
has its own list of permissible values for Class, and
no default is set, so you can ignore this attribute
if you wish.</PARA>
<PARA>An attribute that has the keyword IMPLIED bears no
processing expections if it is absent or its
value is null. Application designers might wish to
supply plausible defaults, but none is specified here.</PARA>
```

```
</DESCRIPTION></HELPTOPIC>
<HELPTOPIC HelpID="">
<HELPLABEL>cptrphrase.gp</HELPLABEL>
<DESCRIPTION>
<PARA>This parameter entity has been introduced to provide
some structure for in-line elements related to computers.
Its contents are: plain text,
Anchor, Comment, Link, ComputerOutput, and UserInput.</PARA>
<PARA>Many of these elements now have attributes
with delimited value lists; some former in-line elements now appear as
values for those attributes.</PARA>
</DESCRIPTION></HELPTOPIC>
<HELPTOPIC HelpID="">
<HELPLABEL>"In-line" vs. "In flow"</HELPLABEL>
<DESCRIPTION>
<PARA>In this document, "in-line" means "occuring within a line
of text, like a character or character string, not causing
a line break." This term is sometimes used to
refer to objects such as an illustration around which
something like a paragraph is wrapped; here that circumstance
will be called "in flow." There is no provision yet
for indicating that an object is in flow, but one could
make creative use of the Role attribute to do so.</PARA>
<PARA>A related point: formal objects have titles; informal
objects do not. That an object is informal does not mean
that it is in-line: these are two different
characteristics.</PARA>
</DESCRIPTION></HELPTOPIC>
<HELPTOPIC HelpID="">
<HELPLABEL>List of Elements/HELPLABEL>
<DESCRIPTION>
<VARIABLELIST>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Actions/EMPHASIS></TERM>
<LISTITEM>
```

```
<PARA>A set of entries, usually in a list form, that comprise
the appropriate set of functions or steps to perform a corrective
action for a situation that is described as part of a help card.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>Anchor</TERM>
<LISTITEM>
<PARA>Marks a target for a Link.
Anchor may appear almost anywhere, and has no content.
Anchor has ID, Pagenum, Remap, Role, and XRefLabel attributes;
the ID is required.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Caution/EMPHASIS></TERM>
<LISTITEM>
<PARA>An admonition set off from the text;
Tip, Warning, Important, and Note all share its model.
Its contents may include paragraphs, lists, and so forth,
but not another admonition.
Caution and its sisters have common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Comment</EMPHASIS></TERM>
<LISTITEM>
<PARA>A remark made within the document file that
is intended for use during interim stages of production.
A Comment should not be displayed to the reader of the
finished, published work. It may appear almost anywhere,
and may contain almost anything
below the Section level. Note that,
unlike an SGML comment, unless you take steps
to suppress it, the Comment element
will be output by an SGML parser
or application. You may wish to do this to display Comments
along with text during the editorial process.</PARA>
</LISTITEM></VARLISTENTRY>
```

```
<VARLISTENTRY>
<TERM>ComputerOutput</TERM>
<LISTITEM>
<PARA>Data presented to the user by
a computer.
It may contain elements from cptrphrase.gp,
and has common and
MoreInfo attributes For the MoreInfo attribute
see <EMPHASIS>Application.</EMPHASIS></PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Copyright</EMPHASIS></TERM>
<LISTITEM>
<PARA>Copyright information about
a document. It consists of one or
more Years followed by any number of Holders.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>Date</TERM>
<LISTITEM>
<PARA>Date of publication or revision.
It contains plain text. (No provision
has been made for representing eras; you could include this
information along with the date data.)</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Description</EMPHASIS></TERM>
<T.TSTTTEM>
<PARA>A part of a HelpTopic element.
Description may contain in-line elements.
The body may be comprised of paragraphs.
It is used to contain the body of text that
is used as a help card.</PARA>
</LISTITEM></VARLISTENTRY>
```

```
<VARLISTENTRY>
<TERM>
<EMPHASIS>DocHelp</EMPHASIS></TERM>
<TITSTTTEM>
<PARA>A collection of help document components.
A DocHelp entry may have a series of HelpTopic(s).
All back matter is optional, and at this time includes
a Glossary.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>DocInfo</EMPHASIS></TERM>
<LISTITEM>
<PARA>Metainformation for a book
component, in which it may appear. Only Title and AuthorGroup
are required. DocInfo may contain, in order:
the required Title, optional TitleAbbrev and
Subtitle, followed by one or more
AuthorGroups, any number of
Abstracts, an optional RevHistory, and any number of
LegalNotices. DocInfo has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>Emphasis</TERM>
<LISTITEM>
<PARA>Provided for use where you would
traditionally use italics
or bold type to emphasize a word or phrase.
It contains plain text and
has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Equation</EMPHASIS></TERM>
<LISTITEM>
<PARA>A titled mathematical equation displayed
on a line by itself, rather than in-line. It has an optional
Title and TitleAbbrev, followed by either
```

```
an InformalEquation or a Graphic (see
<EMPHASIS>Graphic</EMPHASIS>).
Equation has common and Label attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Example/EMPHASIS></TERM>
<LISTITEM>
<PARA>Intended for sections of program source code
that are provided as examples in the text.
It contains a required Title and an
optional TitleAbbrev, followed by one or more block-oriented
elements in any combination. It has common and Label
attributes. A simple Example might contain a Title
and a ProgramListing.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Figure</EMPHASIS></TERM>
<LISTITEM>
<PARA>An illustration.
It must have a Title, and may have a
TitleAbbrev, followed by one or more of
BlockQuote,
InformalEquation, Graphic,
InformalTable, Link, LiteralLayout,
OLink, ProgramListing, Screen, Synopsis, and ULink,
in any order. Figure has common,
Label, and Float attributes; Float indicates
whether the Figure is supposed to be rendered
where convenient (yes) or at
the place it occurs in the text (no, the default). To
reference an external file containing graphical
content use the Graphic element within Figure.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<EMPHASIS>Footnote/EMPHASIS></TERM>
<LISTITEM>
```

```
<PARA>The contents of a footnote, when
the note occurs outside the block-oriented element in
which the FootnoteRef occurs.
(Compare <EMPHASIS>InlineNote.</EMPHASIS>)
The point in the text where the mark for a specific
footnote goes is indicated by FootnoteRef.
Footnote may contain Para, SimPara, BlockQuote, InformalEquation, InformalTable,
Graphic, Synopsis, LiteralLayout, ProgramListing,
Screen, and any kind of list.
It has ID, Label, Lang, Remap, Role, and XRefLabel
attributes; the ID attribute is required, as
a FootnoteRef must point to it.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Glossary</EMPHASIS></TERM>
<LISTITEM>
<PARA>A glossary of terms. Glossary
may occur within a Chapter, Appendix, or Preface,
or may be a book component in its own right.
It contains in order an optional DocInfo, optional
Title, and optional TitleAbbrev, followed by
any number of block-oriented elements, followed by
one or more GlossEntries or one or more GlossDivs.
It has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>GlossDef</EMPHASIS></TERM>
<LISTITEM>
<PARA>The definition attached to a GlossTerm
in a GlossEntry. It may contain Comments, GlossSeeAlsos,
paragraphs, and other block-oriented elements, in
any order; it has common and Subject attributes. The Subject
attribute may hold a list of subject areas (e.g., DCE RPC
General) as keywords.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>GlossEntry</EMPHASIS></TERM>
```

<LISTITEM>

<PARA>An entry in a Glossary.
It contains, in order, a required
GlossTerm, an optional Acronym,
an optional Abbrev, and either a
GlossSee or any number of GlossDefs.
It has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>

<VARLISTENTRY>
<TERM>GlossTerm</TERM>
<LISTITEM>

<PARA>A term in the text of a Chapter (for example) that is
glossed in a Glossary; also used for those terms in GlossEntries, in the
Glossary itself. As you may not want to tag all occurrences
of these words outside of Glossaries, you might consider
GlossTerm, when used outside of Glossaries, to be similar
to FirstTerm, except that GlossTerm may contain other
in-line elements. GlossTerm contains in-line elements
and has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>

<VARLISTENTRY>
<TERM>Graphic</TERM>
<LISTITEM>

<PARA>Encloses graphical data or

points via an attribute to an external file containing such data, and is to be rendered as an object, not in-line. It has Format, Fileref, Entityref, and ID attributes. The format attribute may have the value of any of the formats defined at the head of the DTD, including CGM-CHAR, CGM-CLEAR, DITROFF, DVI, EPS, EQN, FAX, FAXTILE, GIF, IGES, PIC, PS, TBL, TEX, TIFF.</PARA>

<PARA>The value of Fileref should be a filename, qualified by
a pathname if desired; the value of Entityref should be that of an
external data entity. If data is given as the
content of Graphic, both Entityref and Fileref,
if present at all, should
be ignored, but a Format value should be supplied.

```
if no data is given as the content of
Graphic and a value for Entityref
is given, Fileref, if present, should be ignored
but no Format value should be supplied.
Finally, if there is no content for Graphic and
Entityref is absent or null, Fileref must be
given the appropriate value, and again no
Format value should be supplied.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>HelpTopic</EMPHASIS></TERM>
<LISTITEM>
<PARA>A part of a DocHelp document.
HelpTopic contains a HelpLabel, followed in order by
a Description, and optionally an Actions area.
HelpTopic has common and HelpId attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>HelpLabel</EMPHASIS></TERM>
<LISTITEM>
<PARA>The text of a heading or the title of the HelpTopic
block-oriented element. HelpLabel may contain
in-line elements, and has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>InlineEquation</TERM>
<LISTITEM>
<PARA>An untitled mathematical equation
occurring in-line or as the content of an Equation.
It contains a Graphic, and has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>InlineGraphic</TERM>
<LISTITEM>
<PARA>Encloses graphical data or
points via an attribute to an external file containing such data,
```

and is to be rendered in-line. InlineGraphic has Format, Fileref, Entityref, and ID attributes. The format attribute may have the value of any of the formats defined at the head of the DTD, under "Notations." If it is desired to point to an external file, a filename may be supplied as the value of the Fileref attribute, or an external entity name may be supplied as the value of the Entityref attribute.</para> </LISTITEM></VARLISTENTRY> <VARLISTENTRY> <TERM> <EMPHASIS>ItemizedList</EMPHASIS></TERM> <LISTITEM> <PARA>A list in which each item is marked with a bullet, dash, or other dingbat (or no mark at all). It consists of one or more ListItems. A ListItem in an ItemizedList contains paragraphs and other block-oriented elements, which may in turn contain other lists; an ItemizedList may be nested within other lists, too. It has common attributes and a Mark attribute. Your application might supply the mark to be used for an ItemizedList, but you can use this attribute to indicate the mark you desire to be used; there is no fixed list of these.hfill\break <EMPHASIS>Usage Note:</EMPHASIS> You might want to use one of the ISO text entities that designates an appropriate dingbat.</PARA> </LISTITEM></VARLISTENTRY> <VARLISTENTRY> <TERM>Link</TERM> <LISTITEM> <PARA>A hypertext link. At present, all the link types represented in the DTD are provisional. Link is less provisional than the others, however. In HyTime parlance, Link is a clink. It may contain in-line elements and has Endterm, Linkend, and Type attributes. The required Linkend attribute specifies the target of the link, and the optional Endterm attribute specifies text that is to be fetched from elsewhere in the document to appear in the Link. You can also supply this text directly as the content of the Link, <EMPHASIS>in which case the

```
Endterm attribute is to be ignored (new and tentative
rule for this version, comments invited) </EMPHASIS>.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>ListItem</EMPHASIS></TERM>
<LISTITEM>
<PARA>A wrapper for the elements of
items in an ItemizedList or OrderedList; it also
occurs within VarListEntry in VariableList.
It may contain just about anything except Sects and book components.
It has common attributes and an Override attribute, which
may have any of the values of ItemizedList's
Mark attribute; use Override to override the mark
set at the ItemizedList level, when you desire to create
ItemizedLists with varying marks.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>LiteralLayout</EMPHASIS></TERM>
<LISTITEM>
<PARA>The wrapper for lines set off from
the main text that are not tagged as Screens, Examples,
or ProgramListing, in which line breaks and leading
white space are to be regarded as significant.
It contains in-line elements, and has common
and Width attributes, for specifying a number representing
the maximum width of the contents.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Note</EMPHASIS></TERM>
<LISTITEM>
<PARA>A message to the user, set off from the text.
See <EMPHASIS>Caution.</EMPHASIS></PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
```

```
<EMPHASIS>OrderedList</EMPHASIS></TERM>
<LISTITEM>
<PARA>A numbered or lettered list, consisting of
ListItems. A ListItem in an
OrderedList contains paragraphs and other
block-oriented elements, which
may in turn contain other lists; an OrderedList may be
nested within other lists, too.
OrderedList has common attributes, along with
a Numeration attribute, which
may have the value Arabic, Upperalpha, Loweralpha,
Upperroman, or Lowerroman. The default is Arabic (1, 2, 3, . . .).
It has an InheritNum attribute, for which the value Inherit specifies for a
nested list that the numbering of ListItems should include the
number of the item within which they are nested (2a, 2b, etc.,
rather than a, b, etc.); the default value is Ignore.
It has a Continuation attribute, with values
Continues or Restarts (the default), which may be used to
indicate whether the numbering of a list begins afresh (default)
or continues that of the immediately preceding list (Continues).
You need supply the Continuation attribute only
if your list continues the numbering of the preceding list.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Para</EMPHASIS></TERM>
<LISTITEM>
<PARA>A paragraph. A Para may not
have a Title: to attach a Title to a Para use FormalPara.
may contain any in-line element and almost
any block-oriented element. Abstract, AuthorBlurb, Caution,
Important, Note, and Warning are excluded, as are Sects and higher-level
elements. Para has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>ProgramListing</EMPHASIS></TERM>
<LISTITEM>
<PARA>A listing of a program.
Line breaks and leading
```

```
white space are significant in a ProgramListing, which
may contain in-line elements, including LineAnnotations.
(LineAnnotations are a document author's
comments on the code, not the comments written
into the code itself by the code's author.)
ProgramListing has common and Width attributes, the
latter for specifying a number representing the maximum
width of the contents.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>Term</TERM>
<LISTITEM>
<PARA>The hanging term attached to a ListItem
within a VarListEntry in a
VariableList; visually, a VariableList
is a set of Terms with attached items such as paragraphs. Each
ListItem may be associated with a set of Terms. Term may contain
in-line elements. It has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Tip</EMPHASIS></TERM>
<LISTITEM>
<PARA>A suggestion to the user, set off from
the text. See <EMPHASIS>Caution.</EMPHASIS></PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Title</EMPHASIS></TERM>
<LISTITEM>
<PARA>The text of a heading or the title of a
block-oriented element. Title may contain
in-line elements, and has common and PageNum attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>Trademark</TERM>
<LISTITEM>
```

```
<PARA>A trademark. It may contain members of cptrphrase.gp,
and has common and Class attributes.
Class may have the values Service, Trade, Registered,
or Copyright; the default is Trade.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>UserInput</TERM>
<LISTITEM>
<PARA>Data entered by the user.
It may contain elements from cptrphrase.gp,
and has common and MoreInfo attributes. For the MoreInfo attribute
see <EMPHASIS>Application.</EMPHASIS></PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>VariableList</EMPHASIS></TERM>
<LISTITEM>
<PARA>An optionally
titled list of VarListEntries, which are
composed of sets of one or more Terms with associated
ListItems; ListItems contain paragraphs and other block-oriented
elements in any order. Inclusions
are as for OrderedList (see <EMPHASIS>OrderedList</EMPHASIS>).
VariableList has common attributes.</PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>VarListEntry</EMPHASIS></TERM>
<LISTITEM>
<PARA>A component of VariableList (see
<EMPHASIS>VariableList/EMPHASIS>). It has common attributes./PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>
<EMPHASIS>Warning</EMPHASIS></TERM>
<LISTITEM>
<PARA>An admonition set off from the text.
```

```
See <EMPHASIS>Caution.</EMPHASIS></PARA>
</LISTITEM></VARLISTENTRY>
<VARLISTENTRY>
<TERM>XRef</TERM>
<LISTITEM>
<PARA>Cross reference link to another part of the document.
It has Linkend and Endterm attributes, just like Link,
but like Anchor, it may have no content.
XRef must have a Linkend, but the Endterm is optional.
If it is used, the content of the element it points
to is displayed as the text of the cross reference;
if it is absent, the XRefLabel of the cross-referenced
object is displayed. To include in the cross reference
generated text associated with the object referred to,
use your application's style sheet. See <EMPHASIS>Link.</EMPHASIS></PARA>
</LISTITEM></VARLISTENTRY>
</VARIABLELIST>
</DESCRIPTION></HELPTOPIC>
</DOCHELP>
```

An Example of Implementing Help in an Application

This section provides a complete example of help integrated with an application.

Example C-3 lists a C program that implements a Help menu, a **Help** button, and context-sensitive help. You can find this file online at /usr/share/Insight/XHELP/samples/exampleApp/exampleAppXm.c.

Example C-4 lists the help document for exampleAppXm. You can find it online at /usr/share/Insight/XHELP/samples/exampleApp/exampleAppXm.sgm.

Example C-5 lists the helpmap file for exampleAppXm. You can find it online at /usr/share/Insight/XHELP/samples/exampleApp/help/exampleAppXm.helpmap.

Example C-3 An Example of Integrating SGIHelp With an Application

*_____

```
exampleAppXm.c
 * File:
 * Date:
                 3/25/94
 * Compile with: cc -o exampleAppXm exampleAppXm.c -lhelpmsg -lXm -lXt -lX11
 * Purpose:
                 An simple example program that shows how to use the SGI
                 Help system from a Motif application.
                 This program displays a few buttons on a bulletin board
                 alongwith a help menu. The use of context sensitive help
                 is also demonstrated.
 * standard include files
 * /
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <X11/cursorfont.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <Xm/MessageB.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/Separator.h>
 * include for for calling/using SGIHelp
 * /
#include <helpapi/HelpBroker.h>
/*
 * forward declarations of functions
 * /
Widget initMotif(int
                             *argcP,
                                              /* Initializes motif and
```

```
char *argv[], /* and returns the top level*/
                                            /* shell.
                XtAppContext *app_contextP,
                Display **displayP);
                                             /*creates the main window, */
void createInterface(Widget parent);
                                              /*menus, and the buttons
                                                                       * /
                                              /*on the main window
                                                                        * /
                                              /*callbacks for each of
                                                                       * /
void clickForHelpCB();
void overviewCB();
                                              /*the help menu's
                                                                        * /
void taskCB();
void indexCB();
void keysAndShortcutsCB();
void productInfoCB();
void infoDialogCB();
Widget _mainWindow, _infoDialog=NULL;
 * main()
main(int argc, char *argv[])
   Display *display;
   XtAppContext app_context;
   Widget toplevel;
   toplevel = initMotif (&argc,argv,&app_context,&display);
   createInterface(toplevel);
   XtRealizeWidget(toplevel);
   XtAppMainLoop(app_context);
```

```
* Function: initMotif()
 * Purpose: Initializes Motif and creates a top level shell.
             Returns the toplevel shell.
             Makes the call to initialize variables for the SGIHelp
             interface...note that it does not *start* the sqihelp
             process. That is done when a request for help is made,
             if and only if the sgihelp process is not already
             running.
Widget initMotif(int *argcP,char *argv[],XtAppContext *app_contextP,
                 Display **displayP)
{
    Widget toplevel;
   XtToolkitInitialize();
    *app_contextP = XtCreateApplicationContext();
    *displayP = XtOpenDisplay(*app_contextP,NULL,"exampleAppXm",
                                  "exampleAppXmClass", NULL,
                                  0,argcP, argv);
    if (*displayP == NULL) {
        fprintf (stderr, "Could not open display.\n");
        fprintf (stderr, "Check your DISPLAY environment variable.\n");
        fprintf (stderr, "Exiting...\n");
        exit(-1);
    }
    toplevel = XtAppCreateShell("exampleAppXm", NULL,
                                applicationShellWidgetClass,
                                *displayP, NULL,0);
 * initialize variables for SGIHelp
    SGIHelpInit(*displayP, "exampleAppXm", ".");
    return (toplevel);
}
```

```
* Function: createInterface()
void createInterface(Widget parent)
           args[10];
    Arg
    int
           i;
    Widget baseForm;
    Widget menuBar;
    Widget demoLabel, demoButton;
    Widget pulldown1, pulldown2, cascade1, cascade2;
    Widget menuButtons[6];    /*we will create at max 6 buttons on a menu*/
    XmString xmStr;
/*
 * mainWindow is an XmMainWindow
 * on which the whole interface is built
 * /
    i=0;
    _mainWindow = XmCreateMainWindow(parent, "mainWindow", args, i);
    XtManageChild(_mainWindow);
 * baseForm is the workArea for the
 * mainWindow above.
 * /
    i=0;
    XtSetArg (args[i],XmNwidth,400);i++;
    XtSetArg (args[i],XmNheight,300);i++;
    XtSetArg (args[i],XmNverticalSpacing,40);i++;
    baseForm = XmCreateForm(_mainWindow, "baseForm", args,i);
    XtManageChild(baseForm);
 * On this bulletin board, put a label and a button
 * for demonstrating callbacks and context sensitive
 * help.
 * /
```

```
i = 0;
   xmStr = XmStringCreateSimple("SGI Help!");
   XtSetArg (args[i],XmNlabelString,xmStr);i++;
   XtSetArg (args[i],XmNtopAttachment,XmATTACH_FORM);i++;
   XtSetArg (args[i],XmNrightAttachment,XmATTACH_FORM);i++;
   XtSetArg (args[i],XmNleftAttachment,XmATTACH_FORM);i++;
   XtSetArg (args[i],XmNalignment,XmALIGNMENT_CENTER);i++;
   demoLabel = XmCreateLabel(baseForm, "sgiHelpLabel", args, i);
   XtManageChild(demoLabel);
   XmStringFree(xmStr);
   i=0;
   xmStr = XmStringCreateSimple("Click Here For Help");
   XtSetArg (args[i],XmNlabelString,xmStr);i++;
   XtSetArg (args[i],XmNrightAttachment,XmATTACH_FORM);i++;
   XtSetArg (args[i],XmNbottomAttachment,XmATTACH_FORM);i++;
   demoButton = XmCreatePushButton(baseForm, "sgiHelpPushButton", args,i);
   XtManageChild(demoButton);
   XmStringFree(xmStr);
   XtAddCallback(demoButton,XmNactivateCallback,taskCB,NULL);
* build a pulldown menu system, including the "help" menu
  menuBar
             = XmCreateMenuBar(_mainWindow, "menuBar", NULL, 0);
   XtManageChild(menuBar);
   pulldown1 = XmCreatePulldownMenu(menuBar, "pulldown1", NULL, 0);
  pulldown2 = XmCreatePulldownMenu(menuBar, "pulldown2", NULL, 0);
   i=0;
   XtSetArg (args[i],XmNsubMenuId,pulldown1);i++;
   cascade1 = XmCreateCascadeButton(menuBar, "File", args, i);
   XtManageChild(cascade1);
   i=0;
   XtSetArg (args[i],XmNsubMenuId,pulldown2);i++;
   cascade2 = XmCreateCascadeButton(menuBar, "Help", args, i);
   XtManageChild(cascade2);
* Declare this to be the Help menu
* /
   i=0;
   XtSetArg (args[i],XmNmenuHelpWidget,cascade2);i++;
```

```
XtSetValues(menuBar,args,i);
  menuButtons[0] = XmCreatePushButton(pulldown1, "Exit", NULL, 0);
  XtManageChildren(menuButtons,1);
  XtAddCallback(menuButtons[0],XmNactivateCallback,(XtCallbackProc)exit,0);
  menuButtons[0] = XmCreatePushButton(pulldown2, "Click for Help", NULL, 0);
  menuButtons[1] = XmCreatePushButton(pulldown2, "Overview", NULL, 0);
  XtManageChild( XmCreateSeparator(pulldown2, "separator1",NULL,0) );
  menuButtons[2] = XmCreatePushButton(pulldown2, "Sample Help Task", NULL, 0);
  XtManageChild( XmCreateSeparator(pulldown2, "separator2", NULL, 0) );
  menuButtons[3] = XmCreatePushButton(pulldown2, "Index", NULL, 0);
  menuButtons[4] = XmCreatePushButton(pulldown2, "Keys and Shortcuts", NULL, 0);
  XtManageChild( XmCreateSeparator(pulldown2, "separator3",NULL,0) );
  menuButtons[5] = XmCreatePushButton(pulldown2, "Product Information", NULL, 0);
  XtManageChildren(menuButtons,6);
* add callbacks to each of the help menu buttons
  XtAddCallback(menuButtons[0],XmNactivateCallback,clickForHelpCB,NULL);
  XtAddCallback(menuButtons[1],XmNactivateCallback,overviewCB,NULL);
  XtAddCallback(menuButtons[2],XmNactivateCallback,taskCB,NULL);
  XtAddCallback(menuButtons[3],XmNactivateCallback,indexCB,NULL);
  XtAddCallback(menuButtons[4],XmNactivateCallback,keysAndShortcutsCB,NULL);
  XtAddCallback(menuButtons[5],XmNactivateCallback,productInfoCB,NULL);
* set the bulletin board and menubar into
* the main Window.
  XmMainWindowSetAreas(_mainWindow,menuBar,NULL,NULL,NULL,baseForm);
* void clickForHelpCB()
            Provides context-sensitivity within an application;
            makes a request to the sgihelp process.
```

```
void clickForHelpCB(Widget wid, XtPointer clientData, XtPointer callData)
     static Cursor cursor = NULL;
     static char path[512], tmp[512];
    Widget shell, result, w;
     strcpy(path, "");
     strcpy(tmp, "");
 * create a question-mark cursor
     if(!cursor)
         cursor = XCreateFontCursor(XtDisplay(wid), XC_question_arrow);
    XmUpdateDisplay(_mainWindow);
 * get the top-level shell for the window
     shell = _mainWindow;
    while (shell && !XtIsShell(shell)) {
            shell = XtParent(shell);
     }
/*
 * modal interface for selection of a component;
 * returns the widget or gadget that contains the pointer
 * /
    result = XmTrackingLocate(shell, cursor, FALSE);
     if( result ) {
         w = result;
 * get the widget hierarchy; separate with a '.';
 * this also puts them in top-down vs. bottom-up order.
 * /
         do {
              if( XtName(w) ) {
                  strcpy(path, XtName(w));
                  if(strlen(tmp) > 0)
                      strcat(path, ".");
```

```
strcat(path, tmp);
                  strcpy(tmp, path);
              }
              w = XtParent(w);
         } while (w != NULL && w != shell);
          * send msg to the help server-widget hierarchy;
          * provide a mapping to produce the key to be used
          * In this case, we'll let the sgihelp process do
          * the mapping for us, with the use of a helpmap file
          * Note that parameter 2, the book name, can be found
          * from the helpmap file as well. The developer need
          * not hard-code it, if a helpmap file is present for
          * the application.
          * /
          if( strlen(path) > 0 ) {
                SGIHelpMsg(path, NULL, NULL);
 * void overviewCB()
void overviewCB()
* Using the mapping file allows us to specify
* a "Overview" help card for each window in
* our application. In this case, we will point
 * to a specific one. Note that the book name is
 * specified, but not necessary if a helpmap file
 * exists for this application.
```

```
* /
       SGIHelpMsg("overview", "exampleAppXmHelp", NULL);
 * void indexCB()
void indexCB()
* For the index window to work for this application,
* a helpmap file MUST be present!
      SGIHelpIndexMsg("index", NULL);
 * void taskCB()
void taskCB()
* For the task found in the help menu or a pushbutton, we
* use a specific key/book combination.
* /
       SGIHelpMsq("help_task", "exampleAppXmHelp", NULL);
 * void keysAndShortcutsCB()
void keysAndShortcutsCB()
```

```
/*
 * This would point to the help card that contains
 * information about the use of keys/accelerators, etc.
 * for your application.
 * /
        SGIHelpMsg("keys", "exampleAppXmHelp", NULL);
 * void productInfoCB()
void productInfoCB()
 * Pops up a dialog showing product version information.
 * This area has nothing to do with SGIHelp, but is included
 * for completeness.
 biov
          buildInfoDialog();
 XmString xmStr;
 Arg
           args[10];
  int
           i;
    if( _infoDialog == NULL ) {
       buildInfoDialog();
       XtRealizeWidget( _infoDialog );
    xmStr=XmStringCreateSimple("Example Motif App Using SGIHelp version 1.0");
    i=0;
    XtSetArg (args[i],XmNmessageString,xmStr);i++;
    XtSetValues(_infoDialog, args, i);
    XmStringFree(xmStr);
   XtManageChild(_infoDialog);
}
```

```
void buildInfoDialog()
         args[10];
  Arg
  int
         i;
 * Build the informational dialog to display the version info
 * /
    i=0;
    XtSetArg (args[i],XmNautoUnmanage,True);i++;
    XtSetArg (args[i],XmNdialogType,XmDIALOG_WORKING);i++;
    XtSetArg (args[i],XmNdialogStyle,XmDIALOG_APPLICATION_MODAL);i++;
    _infoDialog = XmCreateInformationDialog(_mainWindow, "infoDialog", args, i);
    XtAddCallback(_infoDialog, XmNokCallback, infoDialogCB, NULL);
    XtUnmanageChild(XmMessageBoxGetChild(_infoDialog, XmDIALOG_CANCEL_BUTTON));
    XtUnmanageChild(XmMessageBoxGetChild(_infoDialog, XmDIALOG_HELP_BUTTON));
void infoDialogCB()
    if ( _infoDialog ) {
         XtUnmanageChild(_infoDialog);
         /* Explicitly set the input focus */
         XSetInputFocus(XtDisplay(_mainWindow), PointerRoot,
                        RevertToParent, CurrentTime);
                      Example C-4
                                   Help Source File for Example Program
<dochelp>
<HelpTopic HelpID="overview">
<Helplabel>Example Motif Application Using SGIHelp/Helplabel>
<Description>
<para>
This application is intended to show the developer how
the <glossterm>SGIHelp</glossterm> system can work for you.
It displays (in the included
sample code, exampleAppXm.c) usage of various widgets, a sample
help menu, full-context-sensitivity, and calls to
the <glossterm>SGIHelp</glossterm> server process via the API.
```

```
</para>
<Figure ID="figure_01">
    <title>exampleAppXm Main Window</title>
    <Graphic fileref="mainwnd.gif" format="GIF"></Graphic>
</Figure>
<para>
The application itself is very simple, composed of
a <Link Linkend="ID002">File menu, </Link>
a <Link Linkend="ID003">Help menu,</Link>
a <Link Linkend="ID005">Pushbutton,</Link>
and a <Link Linkend="ID004">Label</Link>.
The user can choose items from the
<Link Linkend="ID003">Help menu</Link> to
contact the <glossterm>SGIHelp</glossterm> server process to
cause different help cards to be rendered.
<para>To quit the application, use the "Exit" command
found under the <Link Linkend="ID002">File menu</Link>.
</Description>
</HelpTopic>
<HelpTopic HelpID="file_menu">
<Helplabel><Anchor Id="ID002">The File Menu</Helplabel>
<Description>
<para>The following items (and their functions) are part of
the File menu:</para>
<VariableList>
<VarListEntry>
<term>Exit</term>
<ListItem><para>Used to quit the exampleAppXm application.</listitem>
</VarListEntry>
</VariableList>
</Description>
</HelpTopic>
<HelpTopic HelpID="help_menu">
<Helplabel><Anchor Id="ID003">The Help Menu</Helplabel>
<Description>
<para>The following items (and their functions) are part of
the Help menu:</para>
```

```
<VariableList>
<VarListEntry>
<term>Click for Help</term>
<ListItem><para>Used to put the application in context sensitive mode.
Will cause the cursor to turn into a "?" at which point the user can
click on any entry in the application's window to obtain help.</para></listitem>
</VarListEntry>
<VarListEntry>
<term>Overview</term>
<ListItem><para>Used to display a help overview card for the current
window.</para></listitem>
</VarListEntry>
<VarListEntry>
<term>Index</term>
<ListItem><para>Used to display from SGIHelp an Index of help topics for
the given application.</para></listitem>
</VarListEntry>
<VarListEntry>
<term>Keys & Shortcuts</term>
<ListItem><para>Used to display a help card that describes any special
key combinations this application uses.</para></listitem>
</VarListEntry>
<VarListEntry>
<term>Product Info</term>
<ListItem><para>Pops up a dialog that displays to the user any version or
copyright information for this application.</para></listitem>
</VarListEntry>
</VariableList>
<para>To access any menu items, click on the menu item
that is a part of the menubar. When the menu pops-up,
highlight the desired item, and release the mouse button.
</para>
</Description>
</HelpTopic>
<HelpTopic HelpID="help_label">
<Helplabel><Anchor Id="ID004">A Label</Helplabel>
<Description>
<para>You have clicked on a Label. It simply displays information
to the user and serves no other useful pourpose.</para>
<tip><para>Basically, a label is useless. For information only.</para></tip>
</Description>
</HelpTopic>
```

```
<HelpTopic HelpID="help_button">
<Helplabel><Anchor Id="ID005">A Pushbutton</Helplabel>
<Description>
<para>You have clicked on a Pushbutton. A pushbutton, when
clicked, will activate some type of command within the application.</para>
</Description>
</HelpTopic>
<HelpTopic HelpID="keys">
<Helplabel><Anchor Id="ID006">Keys and Shortcuts/Helplabel>
<Description>
<para>This card displays all known keys and shortcuts for this
application.</para>
<warning><para>This application has no shortcuts./warning>
</Description>
</HelpTopic>
<HelpTopic HelpID="help_task">
<Helplabel><Anchor Id="ID007">A Sample Help Task</Helplabel>
<Description>
<para>
When creating your application and help text, you may wish
to highlight certain common tasks. This help card was
displayed from either a menu item or a pushbutton.
</para>
<para>
To perform such an operation within your code, the
associated callback that contacts the <glossterm>SGIHelp</glossterm> server
can be constructed as shown below.</para>
<Example>
<Title>Sample Help Task Callback</Title>
<para>
The following is a listing derived from a "C" program:
<ProgramListing>
    /* create menu items, pushbuttons, etc. */
   void taskCB()
     * For the task found in the help menu,
```

```
* we'll use a specific key/book
     * combination.
     * /
        SGIHelpMsg("key", "myBook", NULL);
</ProgramListing>
</para>
</Example>
<para>It's relatively simple process to integrate help
into your application. In fact, the <glossterm>SGIHelp</glossterm>
process only requires <emphasis>two</emphasis> function calls.
</para>
</Description>
</HelpTopic>
<Glossary>
<Title>Glossary</Title>
<GlossEntry>
<GlossTerm>SGIHelp</GlossTerm>
<GlossDef>
<para>This is Silicon Graphics, Inc. version of a "Xhelp" compatible
server. Through the use of an available API, and a help text
compiler, books can be constructed that can be used to render
help information for the given application.</para>
</GlossDef>
</GlossEntry>
</Glossary>
</dochelp>
                      Example C-5
                                    Helpmap for Example Program
1; example AppXmHelp; Example Motif App
Overview;0;overview;exampleAppXm.overview;exampleAppXm.mainWindow.baseForm;exampleAppXm.mainW
indow.menuBar;exampleAppXm.mainWindow
2;exampleAppXmHelp;File Menu;1;file_menu;exampleAppXm.mainWindow.menuBar.File
2;exampleAppXmHelp;Help Menu;1;help_menu;exampleAppXm.mainWindow.menuBar.Help
2; exampleAppXmHelp; A Label Entry; 1; help_label; exampleAppXm.mainWindow.baseForm.sgiHelpLabel
2;exampleAppXmHelp;A Pushbutton
Entry;1;help_button;exampleAppXm.mainWindow.baseForm.sqiHelpPushButton
2; exampleAppXmHelp; Keys and Shortcuts; 0; keys; exampleAppXm.keys
2;exampleAppXmHelp;A Sample Help Task;0;help_task;exampleAppXm.exampleAppXm
```

The Icon Description Language

Use IconSmith to draw your icons. To learn how to use IconSmith, see Chapter 12, "Using IconSmith." After you draw your icon, include it in the FTR file using the ICON rule described in Chapter 13, "File Typing Rules." IconSmith writes the ICON rule for you using the **icon description language**. This appendix describes the icon description language that IconSmith uses to write the ICON rule. This information is provided for completeness. Do not try to write the ICON rule directly in the icon description language.

The icon description language is a restricted subset of the C programming language. It includes line and polygon drawing routines from the IRIS Graphics Library $^{\text{TM}}$ (GL), as well as some additional routines that are not in the GL. The description routine for a given icon is similar in structure to a C subroutine without the subroutine and variable declarations. The valid symbols and functions in the icon description language are described below.

Operators

You can use these C language operators in an icon description routine:

+ -* / & | ^! % = () }

You can use these C language conditional operators in an icon description routine:

&& | | == != < > <=

Constants

You can use these logical constants in an icon description routine:

true false

Variables

The following icon status variables are set by the Desktop. You can use them in an icon description routine:

opened located selected current disabled

These variables have values of either **true** or **false**. You can use them in a conditional statement to alter the appearance of an icon when it has been manipulated in various ways from the Desktop.

You can use other legal C variables in an icon description routine, without a declaration; all variables are represented as type **float**. Any variable name is acceptable, provided it does not collide with any of the predefined constants, variables, or function names in the icon description language.

Functions

The icon description functions comprise, for the most part, a very restricted subset of the C language version of the IRIS Graphics Library, modified for 2-D drawing. See Table D-1 for a list of all the icon description functions.

 Table D-1
 Icon Description Functions

Function	Definition	
arc(x, y, r, startang, endang)	Draw an arc starting at icon coordinates x , y ; with radius r ; starting at angle startang ; ending at angle endang . Angle measures are in tenths of degrees.	
arcf(x, y, r, startang, endang)	Like arc, but filled with the current pen color.	
bclos(color)	Like pclos, but uses color for the border (outline) color of the polygon.	
bgnclosedline()	Begin drawing a closed, unfilled figure drawn in the current pen color. Used in conjunction with vertex and endclosedline.	
bgnline()	Like bgnclosedline, except the figure is not closed. Used in conjunction with vertex and endline.	
bgnoutlinepolygon	Begin drawing a polygon filled with the current pen color. The polygon is outlined with a color specified by endoutlinepolygon. Also used in conjunction with vertex.	
bgnpoint()	Begin drawing a series of unconnected points defined using calls to vertex. Used in conjunction with vertex and endpoint.	
bgnpolygon()	Like bgnoutlinepolygon except the polygon is not outlined. Used conjunction with vertex and endpolygon.	
color(n)	Set current pen color to color index n .	
draw(x, y)	Draw a line in the current color from the current pen location to \mathbf{x} , \mathbf{y} .	
endclosedline()	Finish a closed, unfilled figure started with bgnclosedline.	
endline()	Finish an open, unfilled figure started with bgnline.	
endoutlinepolygon(color)	Finish a filled polygon started with bgnoutlinepolygon and outline it with color .	

Table D-1 (continued)	Icon Description Functions		
Function	Definition		
endpoint()	Finish a series of points started with bgnpoint.		
endpolygon()	Finish a filled, unoutlined polygon started with bgnpolygon.		
for (expr; expr; expr) expr	Note that shorthand operators such as ++ and are not part of the icon description language, so longer hand expressions must be used.		
if (expr) expr [else expr]	Standard C language if-statement.		
include("path")	Tell the Desktop to find the icon geometry in the file with pathname <i>path</i> .		
move(x, y)	Move current pen location to \mathbf{x} , \mathbf{y} .		
pclos()	Draw a line in the current pen color that closes the current polygon, and fill the polygon with the current color.		
pdr(x, y)	Draw the side of a filled polygon in the current pen color, from the current pen location to x , y .		
pmv(x, y)	Begin a filled polygon at location x , y .		
print(expr or "string")	Print the value of the expression expr or string to stdout ; used for debugging.		
vertex(x , y)	Specify a coordinate used for drawing points, lines and polygons by bgnpoint, bgnline, bgnpolygon, and so forth.		

Predefined File Types

This appendix lists the predefined file types and their associated tag numbers that are available for your use. Topics include:

- "Naming Conventions for Predefined File Types"
- "The Predefined File Types and What They Do"

You can use these predefined file types for utilities that do not need a unique, personalized look. You may also want to use these file types as SUPERTYPEs for your own custom file types.

Naming Conventions for Predefined File Types

The file types listed in this appendix are named according to the conventions listed in Table E-1.

Table E-1 Predefined File Type Naming Conventions

If the file type name includes:	Then
1-Narg	it requires at least one argument
1arg	it requires exactly one argument
2arg	it requires exactly two arguments
3arg	it requires exactly three arguments

In all cases, if the expected number of arguments is not received, launch is run so that users can type in the desired options. For more information on the launch command, see the launch(1) reference page.

The Predefined File Types and What They Do

In this section, file types that are essentially the same, except for the number of arguments they require, are grouped together by the "base" file type name, meaning the file type name without the argument codes described in "Naming Conventions for Predefined File Types" on page 309.

For example, to find the file type named "ttyLaunchOut1argExecutable," look under "ttyLaunchOutExecutable." These two file types are identical, except that "ttyLaunchOut1argExecutable" requires exactly one argument.

SpecialFile

"SpecialFile" is a predefined SUPERTYPE, not an actual file type. When you include the SPECIALFILE rule in your file type, you should also declare the "SpecialFile" SUPERTYPE. This allows applications to use isSuper(1) to test whether your file type is a SPECIALFILE.

Directory

```
TYPE Directory
MATCH (mode & 0170000) == 040000;
```

The "Directory" type. Any custom file types you define for directories should include "Directory" as a SUPERTYPE. "Directory" is defined in /usr/lib/filetype/default/sgidefault.ftr.

Ascii

TYPE Ascii

"Ascii" is a pseudotype defined to support routeprint conversions. Actual ASCII text files have the type "AsciiTestFile":

TYPE AsciiTextFile MATCH ascii;

"Ascii" is defined in /usr/lib/filetype/system/sgisystem.converts.ftr and "AsciiTextFile" is defined in /usr/lib/filetype/default/sgidefault.ftr. The Ascii icon is

/usr/lib/filetype/default/iconlib/AsciiText.closed.fti superimposed over /usr/lib/filetype/iconlib/generic.doc.fti.

Source Files

TYPE SourceFile

"SourceFile" is a pseudotype defined to support routeprint conversions. Actual source files have more specific types such as:

```
TYPE Makefile

MATCH (glob("[mM]akefile") || glob("*.mk")) && ascii;

TYPE HeaderFile

MATCH glob("*.h") && ascii;

TYPE CPlusPlusProgram

MATCH glob("*.c++") && ascii;

TYPE CProgram

MATCH glob("*.c") && ascii;

TYPE Program

MATCH (glob("*.c") && ascii;
```

"SourceFile" is defined in

/usr/lib/filetype/system/sgisystem.converts.ftr and the specific types shown above are defined in /usr/lib/filetype/system/sgisystem.ftr.

Binary

"Binary" is a predefined SUPERTYPE, not an actual file type. You can create custom file types using "Binary" as a SUPERTYPE.

ImageFile

```
TYPE ImageFile
```

"ImageFile" is a top-level image pseudotype. You can create custom file types using ImageFile as a SUPERTYPE, or you can use a more specific file type such as:

```
TYPE SGIImage
MATCH short(0) == 000732 ||
# normal SGI image
short(0) == 0155001;
#byte-swapped SGI image
TYPE TIFFImage
MATCH long(0) == 0x49492a00 \mid long(0) == 0x4d4d002a;
# TIFF image
TYPE FITImage
MATCH string(0,2) == "IT";
# FIT image
TYPE PCDimage
MATCH string(2048,7) == "PCD_IPI";
# Kodak Photo CD image pack
TYPE PCD0image
MATCH string(0,7) == "PCD_OPA";
# Kodak Photo CD overview pack
TYPE GIF87Image
MATCH string(0,6) == "GIF87a";
# GIF image (GIF87a format)
TYPE GIF89Image
MATCH string(0,6) == "GIF89a";
# GIF image (GIF89a format)
```

These and other file types are defined in

/usr/lib/filetype/system/sgiimage.ftr.

Executable

"Executable" is a predefined SUPERTYPE, not an actual file type. You can create custom file types using "Executable" as a SUPERTYPE.

Scripts

```
TYPE Script
MATCH (mode & 0111) && ascii;

This is the file type for shell scripts, defined in
/usr/lib/filetype/default/sgidefault.ftr.
```

GenericWindowedExecutable

```
TYPE GenericWindowedExecutable
MATCH tag == 0x000000000;

TYPE Generic1-NargExecutable
MATCH tag == 0x000000020;

TYPE GenericlargExecutable
MATCH tag == 0x000000001;

TYPE Generic2argExecutable
MATCH tag == 0x000000002;

TYPE Generic3argExecutable
MATCH tag == 0x000000003;
```

Simply runs the command. No output or terminal emulation windows are used. These file types are defined in /usr/lib/filetype/system/sgicmds.ftr.

LaunchExecutable

```
TYPE LaunchExecutable

MATCH tag == 0x00000100;

TYPE Launch1-NargExecutable

MATCH tag == 0x00000120;

TYPE Launch1argExecutable

MATCH tag == 0x00000101;

TYPE Launch2argExecutable

MATCH tag == 0x00000102;
```

Same as "GenericWindowedExecutable," except that it runs launch to allow user to enter options prior to running the command. These file types are defined in /usr/lib/filetype/system/sgicmds.ftr.

ttyExecutable

```
TYPE ttyExecutable
MATCH (tag == 0x00000400) || (tag == 0x00000410);
TYPE tty1-NargExecutable
MATCH tag == 0x00000420;
TYPE tty2argExecutable
MATCH tag == 0x00000402;
```

Runs the command in a window that allows terminal I/O. The output window (which is where the terminal emulation is being done) exits immediately upon termination of the command. These file types are defined in

/usr/lib/filetype/system/sgicmds.ftr.

ttyLaunchExecutable

```
TYPE ttyLaunchExecutable
MATCH tag == 0x00000500;

TYPE ttyLaunchl-NargExecutable
MATCH tag == 0x00000520;

TYPE ttyLaunchlargExecutable
MATCH tag == 0x00000501;
```

Same as "ttyExecutable," except that it runs launch to allow user to enter options before running the command. These file types are defined in

/usr/lib/filetype/system/sgicmds.ftr.

ttyOutExecutable

```
TYPE ttyOutExecutable
MATCH (tag == 0x00000600) || (tag == 0x00000610);

TYPE ttyOutl-NargExecutable
MATCH tag == 0x00000620;

TYPE ttyOutlargExecutable
MATCH tag == 0x00000601;

TYPE ttyOut2argExecutable
MATCH tag == 0x00000602;
```

Same as "ttyExecutable," except that the output window persists until the user explicitly dismisses it. These file types are defined in

/usr/lib/filetype/system/sgicmds.ftr.

ttyLaunchOutExecutable

```
TYPE ttyLaunchOutExecutable

MATCH (tag == 0x00000700) || (tag == 0x00000710);

TYPE ttyLaunchOut1-NargExecutable

MATCH tag == 0x00000720;

TYPE ttyLaunchOutlargExecutable

MATCH tag == 0x00000701;

TYPE ttyLaunchOut2argExecutable

MATCH tag == 0x00000702;

TYPE ttyLaunchOut3argExecutable

MATCH tag == 0x00000703
```

Same as "ttyOutExecutable," except that it runs launch to allow user to enter options before running the command. These file types are defined in

/usr/lib/filetype/system/sgicmds.ftr.

FTR File Directories

There are four possible files in which Desktop file types are defined. They are listed here in the order the Desktop scans them:

- 1. /usr/lib/filetype/local
- /usr/lib/filetype/install
- /usr/lib/filetype/system
- 4. /usr/lib/filetype/default

These files are listed in order of precedence. For example, a file type defined in the /usr/lib/filetype/install directory overrides a file type of the same name in the /usr/lib/filetype/system and /usr/lib/filetype/default directories.

In particular, Silicon Graphics uses the /usr/lib/filetype/system and /usr/lib/filetype/default directories to define and maintain system standards. Be especially careful not to override important defaults set in these directories.

Using GoldenGate Data Conversion Services

This appendix describes how to use the GoldenGate data conversion services. It covers these topics:

- "Converting Data Using the GoldenGate Data Conversion Service" on page 319 explains how to use the converters provided.
- "Compiling and Linking Your Program with GoldenGate" on page 338 describes the header file to use when compiling and linking a program.
- "Writing Converters for the GoldenGate Data Conversion Service" on page 339 explains how to customize your own converters.

Converting Data Using the GoldenGate Data Conversion Service

This section describes how you can use the GoldenGate data conversion service in your application. Specifically, it explains:

- "Overview of the Conversion Process" on page 320 describes the steps involved in converting data using GoldenGate.
- "Selecting a Converter" on page 322 describes how to select a converter by querying the converter registry and setting up the conversion context.
- "Using GoldenGate to Convert Data" on page 328 describes how to initialize the conversion pipeline, send data through it, and clean up after the conversion.

Overview of the Conversion Process

To convert data using GoldenGate, follow these steps:

- 1. Choose a converter.
 - Obtain a list of converters that read the source format and write the target format.
 - Create a conversion context structure and set conversion parameters.
 - Evaluate the list of converters to determine which one is best suited for the current conversion.
- 2. Convert data.

There are two methods of converting data, depending on whether the data is in a stream or in a file.

Converting Stream Data

- Initialize the selected converter.
- Send data through the converter and read results back.
- Clean up resources by destroying the conversion context.

Converting Data Files

- Call the file conversion function.
- Clean up resources by destroying the conversion context.

The Converter Registry

GoldenGate maintains a list of available converters in the **converter registry**. This registry contains an entry for each converter, specifying characteristics such as the type of input data it takes and the type of output data it produces. To find out if there are any converters that will convert from format "A" to format "B," you can query the registry.

GoldenGate returns a list of converters that take the specified input and produce the specified output. You can be as specific as you like when querying the registry, to ensure that only relevant converters are listed. You should also use the query to eliminate inappropriate categories of converter, such as those of type StreamToStream if you are converting a file. If the list contains more than one converter, you may need to evaluate the converters to see which one best meets your needs. Even if the list contains only one converter, you should evaluate it to make sure it can handle your conversion request.

Creating a Conversion Context

To communicate with a converter, you must create a **conversion context**. The conversion context is a data object that stores conversion parameters. The conversion context is passed to subsequent library calls that set input and output parameters, evaluate converters, initialize the conversion pipeline, and move data through it.

Once you have created a conversion context and specified the desired conversion parameters, you can evaluate the list of converters you obtained when you queried the registry. For example, suppose you want to convert from one audio format to another and change the sample rate at the same time. Querying the registry returns a list of converters that will convert between the specified input and output formats. To determine if any of these converters will perform the desired sample rate conversion, you have to create a conversion context, set the desired parameters (including input and output sample rate) and then evaluate the individual converters.

Evaluating a Converter

It's best to evaluate a converter before you invoke it to perform a conversion. You do this for the following reasons:

- Evaluation gives the converter an opportunity to inspect your data parameters.
 Some converters will have more functionality than others, even though their input and output types are the same. A well-designed converter will know just by looking at parameters whether it can do the conversion.
- Conversion is typically an expensive operation. If your attempt to convert fails, you
 can still choose a different converter and try again, but you could have avoided lost
 time by trying a converter that can accept your specific request.

Depending on your needs, you can select the first converter on the list that passes the evaluation stage, or evaluate the whole list and use your own rules to choose between those that pass.

Converting Data In a File or Stream

Once you determine the converter to use, the final stage depends on whether you are converting data in a file or a stream.

If you are converting a stream, initialize a conversion pipeline that reads your stream and passes back results as they are available. Then you send all your data through the pipeline and read the results until you see the end of stream marker for the pipeline. At this point, terminate the pipeline. This causes GoldenGate to clean up data structures it keeps for maintaining a stream conversion.

If you are converting a file, the procedure is simpler. You call a single GoldenGate function to perform the operation, and wait for results. If necessary you can provide a callback function that will notify you when results become available. This allows you to service other events going on in your application during what may be a long conversion.

Selecting a Converter

This section describes how to select a converter by querying the converter registry and setting up the conversion context. Specifically, this section covers:

- "Querying the Converter Registry" on page 323, which explains how to obtain a list of possible converters.
- "Setting Up the Conversion Context" on page 326, which describes how to create a conversion context.
- "Evaluating Converters" on page 327, which explains how to find a converter that performs the specified conversion.
- "Getting Converter Details" on page 328, which describes how to get a description
 of a converter.
- "Converter Return Status Values" on page 334, which lists return status values.

Querying the Converter Registry

To query the converter registry, you specify a set of constraints. Each constraint consists of an attribute (such as input format), a value for the attribute, and a comparison operator. For example, you can ask for a converter that has input format equal to "AIFF_FILE," and version number greater than 2. Use the SgCvtSetQueryConstraint() function to fill in an array of SgCvtQueryConstraint structures, then pass the array to the SgCvtQueryRegistry() function. The following code fragment demonstrates a simple query that locates converters capable of converting AIFF_FILE to WAVE_FILE:

The SgCvtQueryRegistry() function returns an array of **converter IDs** that can be used to identify the individual converters.

The *registry* argument specifies the GoldenGate converter registry to be queried. During this call, the registry is located on disk (/var/GoldenGate/ConverterRegistry by default), and its contents parsed to find a converter that matches your requirements.

The first time you call SgCvtQueryRegistry, specify registry as NULL as in the previous example, which causes this lookup. When you have finished converting, you can either call SgCvtFreeRegistry to release the resources that GoldenGate may have cached after reading the file, or you can re-use the value returned in registry for subsequent queries, avoiding the overhead of looking up the file.

If you choose to free the registry between queries, your program will always have the latest information, even if the registry changes while your program is running. If you choose to re-use the registry handle, you have no control over whether or not GoldenGate will re-parse the registry. It will try to use its cache first. If for any reason the cache is invalid, GoldenGate may at its discretion rebuild it by reading the disk-based registry again.

The converters argument returns an array of matching converter IDs, of which the first *num_converters* are valid and matched the query. You should free this array when you are finished using it, using *free*(3).

Table G-1 lists the attributes you can query.

Table G-1 Converter Attributes

Attribute Name	Description
SG_CVT_ATTR_NAME	Converter name
SG_CVT_ATTR_INPUT_FORMAT	Input format
SG_CVT_ATTR_OUTPUT_FORMAT	Output format
SG_CVT_ATTR_IO_METHOD	StreamToStream or FileToFile
SG_CVT_ATTR_INPUT_LABEL	Input format, human readable version
SG_CVT_ATTR_OUTPUT_LABEL	Output format, human readable version
SG_CVT_ATTR_VENDOR	Vendor's name
SG_CVT_ATTR_VERSION	Vendor's version information
SG_CVT_ATTR_DESCRIPTION	Description of converter

Most of the time, you'll be interested in the input format and output format attributes. "Supported Target Formats" on page 101 lists common data formats. Other attributes may be useful when listing converters for users. For example, if you want the user to choose between two converters that perform the same conversion, you can display the vendor names and version numbers.

Table G-2 lists the operators you can use in your query.

Table G-2 Query Operators

Operator	Symbol
equal to	SG_CVT_OP_EQ
not equal to	SG_CVT_OP_NE
less than	SG_CVT_OP_LT
less than or equal to	SG_CVT_OP_LE
greater than	SG_CVT_OP_GT
greater than or equal to	SG_CVT_OP_GE

Note that if more than one constraint is specified on a single attribute, a logical AND is implied. For example, you can select a range of version numbers by setting "version greater than or equal to one" as one constraint and "version less than or equal to three" as a second constraint.

SgCvtQueryConstraint can return the following status value:

SG_CVT_E_SUCCESS

The operation succeeded.

SgCvtQueryRegistry can return the following status values:

SG_CVT_E_SUCCESS

The operation succeeded.

SG_CVT_E_FAILURE

Could not find the registry, or failed to parse it. Most likely when the default registry has been edited to add new converters, and a syntax error introduced. You may also be loading the wrong file. Make sure that if there is a file called ConverterRegistry on your path, it is a valid registry using the CDF syntax. Also make sure the CVT_REGISTRY_OVERRIDE variable is not set.

Setting Up the Conversion Context

Before you can evaluate or use a converter, you must create a conversion context and set parameters governing the conversion. Use the SgCvtCreateConversionContext() function to create a conversion context:

```
SgCvtStatus
SgCvtCreateConversionContext(SgCvtConversionContext *context)
```

SgCvtCreateConversionContext can return the following status values:

```
SG_CVT_E_SUCCESS
```

The operation succeeded.

SG_CVT_E_NOMEM

Insufficient memory to allocate a context.

Next, set any digital media parameters that affect your conversion by calling SgCvtSetContextInfo.

```
SgCvtStatus SgCvtSetContextInfo
   (
   SgCvtConversionContext context,
   unsigned long valuemask,
   SgCvtContextInfo *context_data
   );
```

where

context specifies the context you created with

SgCvtCreateConversionContext

valuemask specifies which fields in the SgCvtContextInfo structure are being set

in the context. This is specified as any of the following OR'ed together:

SG_CVT_INFO_INPUT_PARAMS
SG_CVT_INFO_OUTPUT_PARAMS
SG_CVT_INFO_META_PARAMS
SG_CVT_INFO_INPUT_FILE
SG_CVT_INFO_INPUT_HOST
SG_CVT_INFO_OUTPUT_FILE
SG_CVT_INFO_OUTPUT_HOST

context_data specifies the values being set

SgCvtSetContextInfo can return the following status value:

SG_CVT_E_SUCCESS

The operation succeeded.

See the IRIS Media Libraries Programming Guide for information on setting DMparams.

Evaluating Converters

To evaluate a converter, call SgCvtEvaluateConverter():

SqCvtStatus

SgCvtEvaluateConverter(SgCvtConverterId converter_id,

SgCvtConversionContext context
DMparams **output_params)

where

converter_id is a converter ID returned by the SgCvtQueryRegistry() function

context is a valid conversion context obtained from

SgCvtCreateConversionContext()

output_params returns the output of the request. Converters may set these parameters,

even though they accept the request.

SgCvtEvaluateConverter() can return the following status values:

SG CVT E ACCEPT

The converter can perform the conversion specified by the conversion

context.

SG_CVT_E_REJECT

The converter can't perform the requested conversion.

When evaluating a converter returns a status of SG_CVT_E_ACCEPT, you should take one final step before calling the converter. You should inspect the *output_params* argument, which returns a DMparams list describing the result that the converter will produce. If your program has very strict requirements, this will help protect you if the converter has accepted the request but cannot honor what it considers a minor parameter, or if you passed a parameter it could not understand.

Getting Converter Details

If your program needs to display information about available converters, or do other processing based on the data stored about a converter in the converter registry, call SgCvtGetConverterAttributes() to get a description of it. The function prototype for SgCvtGetConverterAttributes() is shown below.

```
SgCvtStatus
SgCvtGetConverterAttributes(SgCvtConverterId converter_id,
unsigned long converter_attr_mask,
SqCvtConverterAttrs *attributes)
```

When you are finished using the fields of the SgCvtConverterAttrs structure, you should free the string attributes and the structure itself (if you allocated it dynamically) using free(3C).

SgCvtGetConverterAttributes can return the following status value:

```
SG_CVT_E_SUCCESS
```

The operation succeeded.

Using GoldenGate to Convert Data

This section describes the different methods you can use to convert data. Topics include:

- "Converting Data Using File Converters"
- "Converting Data Using Stream Data Converters"

Converting Data Using File Converters

Your file-based data is always converted using the function SgCvtConvertFileToFile. Before you call it however, you need to decide whether you want the function to block while the conversion is going on, or return immediately and let you know later that the conversion is complete.

In many cases blocking mode is sufficient, and it is much simpler to use if your program is not naturally event driven. However, if your application has a GUI, you may prefer non-blocking mode because it allows your event loop to keep running while conversion is going on. When conversion is complete, you are notified through a callback function that you supply, and you can use the converted data.

Both modes are invoked using SgCvtConvertFileToFile:

```
typedef void (*SgCvtCallback)(SgCvtConversionContext context,
                               void *client_data,
                               void *callback_data);
SgCvtStatus SgCvtConvertFileToFile
         SgCvtConversionContext
                                   context,
         SgCvtConverterId
                                   converter_id,
                                   *input_file,
         char
         char
                                   *output_file,
         unsigned long
                                  callback_mask,
         SqCvtCallback
                                  callback,
                                   *client_data
         void
         );
```

where

context the conversion context, holding the I/O filenames and parameters

converter_id the converter ID, returned by SgCvtQueryConverter
input_file pathname of input file. You must have read permission.

output_file pathname of output file. You must have write permission.

callback_mask mask indicating when callback should be called. It should be some

logical combination of the following values:

callback specifies the callback function

client_data a pointer to application-defined data structure that will be passed to the

callback when invoked

If specified, the callback argument is the address of the function to call when conditions specified by the *callback_mask* arise. If the callback function is not specified, or the mask is zero, the function executes in blocking mode.

SgCvtConvertFileToFile can return the following status values:

SG CVT E SUCCESS

The operation succeeded.

SG CVT E BAD CONVERTER TYPE

The converter was not registered as FileToFile IO method.

SG_CVT_E_READ_FAILED

The input file could not be read. It may be missing, or the permissions are insufficient for reading.

SG CVT E WRITE FAILED

The output file could not be written. This can happen if the user does not have write permission for the target directory, or if the supplied pathname was invalid.

Converting Data Using Stream Data Converters

To convert data using your specified converter, you must initialize the conversion pipeline, and then send the data through. After reading the last block of converted data, clean up by destroying the conversion context to free the resources associated with the pipeline. This section covers the following topics:

- "Initializing the Pipeline"
- "Sending and Receiving Data"
- "Cleaning Up"

Initializing the Pipeline

Prepare the converter to receive data by calling SgCvtInitializePipeline():

```
SgCvtStatus
SgCvtInitializePipeline(SgCvtConversionContext context,
SgCvtConverterId converter_id)
```

where

context is a valid conversion context obtained from

SgCvtCreateConversionContext()

converter_id is a converter ID returned by the SgCvtQueryRegistry() function

SgCvtInitializePipeline can return the following status values:

SG CVT E SUCCESS

The operation succeeded.

SG CVT E FAILURE

The context or its contents is bad or one of the subprocesses required to host a converter function could not be launched.

SG_CVT_E_BAD_CONVERTER_TYPE

The converter was not registered as StreamToStream. Converters that are designed to work with streaming data advertise themselves as using the StreamToStream method of I/O in the registry.

Sending and Receiving Data

You may send and receive arbitrarily sized blocks of data, so use a block size that is convenient.

Send data to the converter using SgCvtSendData(). The function prototype for SgCvtSendData() is shown below:

```
SgCvtStatus SgCvtSendData(
SgCvtConversionContext context,
void *data,
size_t length,
DMparams *params,
boolean_t canwait
)
```

where

context is a valid conversion context

data is a pointer to the data block to be converted

length is the length of the data block

params is a DMparams structure describing the data to be converted

canwait is a boolean value that indicates what the function should do if it cannot

send the data immediately. If you specify B_TRUE, SgCvtSendData() will block until it can send the data to the conversion pipeline. If you specify B_FALSE, SgCvtGetData() will return immediately with a status of SG_CVT_E_AGAIN. This status indicates that you should try

again.

SgCvtSendData can return with the following status values:

SG CVT E SUCCESS

The operation succeeded.

SG_CVT_E_FAILURE

An I/O error occurred while trying to send data through the pipe connecting two pipeline components.

SG_CVT_E_AGAIN

Required resources were temporarily unavailable. The caller should retry later.

Read data from the converter using SgCvtGetData(). The function prototype for SgCvtGetData() is shown below:

```
SgCvtStatus SgCvtGetData(
   SgCvtConversionContext context,
   size_t buf_len,
   void *buffer,
   size_t *length_returned,
   DMparams **params_returned,
   boolean_t canwait
)
```

where

context is a valid conversion contextbuf_len specifies the size of buffer

buffer is a pointer to a pre-allocated buffer of at least buf_len bytes.

length_returned is the actual length of the returned data (this may be less than

bytes_requested if non-blocking mode is specified, or if the converter

encounters end-of-stream) SgCvtGetData

params_returned

is a DMparams structure describing the converted data.

canwait

is a boolean value that indicates what the function should do if no data is available. If you specify B_TRUE, SgCvtGetData() will block until data becomes available from the conversion pipeline. If you specify B_FALSE, SgCvtGetData() will return immediately with a status of SG_CVT_E_QUEUE_EMPTY. This status indicates that you should try again.

SgCvtGetData can return the following status values:

SG CVT E SUCCESS

The operation succeeded.

SG CVT E FAILURE

An I/O error occurred while trying to read data from the pipe connecting two pipeline components.

SG_CVT_E_AGAIN

Required resources were temporarily unavailable. The caller should retry later.

SG_CVT_E_END_OF_STREAM

The operation succeeded, and the end of the data has been reached.

The non-blocking mode of SgCvtSendData() and SgCvtGetData() allows programs to continue working on other tasks (such as handling events from a graphical interface) while waiting to send data to or read data from the conversion pipeline.

Cleaning Up

When you've sent the last of the data to the converter, call SgCvtSendEndOfStream() to indicate the end of the data. After you've read the last of the converted data, free the resources associated with the conversion context by calling

```
SgCvtDestroyConversionContext():
SgCvtStatus SgCvtSendEndOfStream(
        SgCvtConversionContext context
);
SgCvtStatus SgCvtDestroyConversionContext(
        SgCvtConversionContext context
)
```

If you need to terminate the conversion process before reaching the end of the data, call SgCvtDestroyConversionContext().

SgCvtSendEndOfStream can return the following status value:

SG CVT E SUCCESS

The operation succeeded.

 ${\tt SgCvtDestroyConversionContext}\ can\ return\ the\ following\ status\ value:$

SG_CVT_E_SUCCESS

The operation succeeded.

Converter Return Status Values

Table G-3 lists converter functions and their return status values.

 Table G-3
 Converter Return Status Values

Function	Return Value	Description
SgCvtSetQueryConstraint	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtQueryRegistry	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_FAILURE	Could not find the registry, or failed to parse it. Most likely when the default registry has been edited to add new converters, and a syntax error introduced. You may also be loading the wrong file. Make sure that if there is a file called ConverterRegistry on your path, it is a valid registry using the converter description file syntax. Also make sure the CVT_REGISTRY_OVERRIDE variable is not set.
SgCvtGetConverterAttributes	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtCreateConversionContext	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_NOMEM	Insufficient memory to allocate a context.
SgCvtDestroyConversionContext	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtSetContextInfo	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtGetContextInfo	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtEvaluateConverter	SG_CVT_E_ACCEPT	The converter can perform the requested conversion.

 Table G-3 (continued)
 Converter Return Status Values

Function	Return Value	Description
	SG_CVT_E_REJECT	The converter cannot perform the requested conversion.
SgCvtInitializePipeline	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_FAILURE	The context or its contents is bad or one of the subprocesses required to host a converter function could not be launched.
	SG_CVT_E_BAD_ CONVERTER_TYPE	The converter was not registered as StreamToStream. Converters that are designed to work with streaming data advertise themselves as using the StreamToStream method of I/O in the registry.
SgCvtTerminatePipeline	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtSendData	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_FAILURE	An I/O error occurred while trying to send data through the pipe connecting two pipeline components.
	SG_CVT_E_AGAIN	The required resources were temporarily unavailable. The caller should retry later.
SgCvtGetData	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_FAILURE	An I/O error occurred while trying to read data from the pipe connecting two pipeline components.
	SG_CVT_E_AGAIN	The required resources were temporarily unavailable. The caller should retry later.

 Table G-3 (continued)
 Converter Return Status Values

Function	Return Value	Description
	SG_CVT_E_END_OF_ STREAM	The operation succeeded, and the end of the data has been reached.
SgCvtSendEndOfStream	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtEncodeParams	SG_CVT_E_SUCCESS	The operation succeeded.
SgCvtDecodeParams	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_NOMEM	Insufficient memory to allocate structures.
SgCvtFree Encoded Params	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_FAILURE	The data could not be decoded.
SgCvtConvertFileToFile	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_BAD_ CONVERTER_TYPE	The converter was not registered as FileToFile I/O method.
	SG_CVT_E_READ_ FAILED	The input file could not be read. It may be missing, or the permissions are insufficient for reading.
	SG_CVT_E_WRITE_ FAILED	The output file could not be written. This can happen if the user does not have write permission for the target directory, or if the supplied pathname was invalid.
SgCvtGetFileSelectionTarget	SG_CVT_E_SUCCESS	The operation succeeded.
	SG_CVT_E_UNKNOWN	The file type could not be
	_ TYPE	determined.
	SG_CVT_E_NO_TARGE T	The selection target for the type of file could not be determined, or there is none.

 Table G-3 (continued)
 Converter Return Status Values

Function	Return Value	Description
	SG_CVT_E_FAILURE	The operation could not be performed for another reason, such as the underlying file typing database library could not be accessed, or the database itself was corrupt or missing.
SgCvtIsPipeline	B_TRUE	The translator is a multi-stage pipeline.
	B_FALSE	The translator is a single-stage converter.

Compiling and Linking Your Program with GoldenGate

To compile and link your program, you need to include the header file SgCvt.h and include the library libcvt in your link line.

An example of a simple GoldenGate program follows. It includes the required header file, enumerates the registered converters, and prints their input and output labels.

```
#include <SgCvt.h>
main(int argc, char **argv)
    SgCvtRegistry registry = NULL;
                        n=0;
    SgCvtStatus
                         s;
    SgCvtConverterId *cvtrs;
    int
                         ncvtrs;
    s = SgCvtQueryRegistry(NULL, 0, &registry, &cvtrs, &ncvtrs);
    for (n=0; n<ncvtrs; n++) {
            SgCvtConverterAttrs attrs;
            SgCvtGetConverterAttributes(cvtrs[n],
                SG_CVT_ATTR_FLAG_INPUT_LABEL
                SG_CVT_ATTR_FLAG_OUTPUT_LABEL,
                &attrs);
           printf("%d %25s -> %s\n",
                n+1, attrs.input_label, attrs.output_label);
            free(attrs.input_label);
            free(attrs.output_label);
     SgCvtFreeRegistry(registry);
}
```

The following Makefile illustrates the compile and link requirements for this program.

```
# # Makefile for GoldenGate Listing sample program
#
CC = cc
TARGET = gg_listing
SOURCES = $(TARGET).

INCLUDES = -I/usr/include/convert

REQLIBS = -lcvt
all:
    $(CC) -o $(TARGET) $(INCLUDES) $(SOURCES) $(REQLIBS)
```

Writing Converters for the GoldenGate Data Conversion Service

This section describes how to write converters that can integrate with GoldenGate and become available to any component that is aware of GoldenGate. The following information assumes that you are familiar with the interfaces described in "Converting Data Using the GoldenGate Data Conversion Service" on page 319. Both converters and applications use many of the functions and data structures.

Overview of the Converter Writing Process

Creating a GoldenGate data converter involves writing the converter and building the DSO, then testing, registering, and installing the converter. The topics below describe:

- "Writing Converter Code" on page 340 explains how to write the code that converts
 the data, or choose an existing command that you want to make available through
 the conversion service.
- "Building a DSO" on page 347 describes how to create a Dynamic Shared Object (DSO) and write a registry entry using converter description file syntax.
- "Testing Your Converter" on page 351 explains how to test your converter.
- "Registering Your Converter" on page 352 describes how to register your converter to make it available to GoldenGate clients.
- "Installing Your Converter" on page 353 lists the library location for converter DSOs.

[&]quot;Some Sample Converters" on page 353 shows annotated code for two converters.

Writing Converter Code

This section describes how to write converter code and includes the following topics:

- "Implementing Your Converter Handling Evaluation Requests"
- "Implementing Your Converter Handling Conversion Requests"
- "Input and Output Formats"
- "Process Blocking"
- "Programming Constraints"
- "Example of a Simple Stream Converter"

Implementing Your Converter - Handling Evaluation Requests

When the operation field of the SgCvtConverterData structure passed to your converter is equal to SG_CVT_REQ_EVALUATE, your converter should inspect the input, output, and meta parameters held in the conversion context and determine whether or not it can satisfy the request, without actually performing conversion.

If your converter can satisfy the request, it should set the status_return field of the SgCvtConverterData structure to SG_CVT_E_ACCEPT before returning. Otherwise it should set status_return to SG_CVT_E_REJECT.

Implementing Your Converter - Handling Conversion Requests

When the operation field of the SgCvtConverterData structure is equal to SG_CVT_REQ_CONVERT, your converter must extract the necessary information from the SgCvtConverterData structure it is passed, and perform the conversion if possible. If conversion is successful, it should return with the status_return field set to SG_CVT_E_SUCCESS, and if it is unsuccessful, the status_return field should be set to either SG_CVT_E_FAILURE or a more specific error code if appropriate (see the error codes available in SgCvt.h).

How the converter is implemented depends on whether you are writing the conversion code yourself, or simply using an existing command-line converter.

If you are creating a "wrapper" to make an existing UNIX command available through the GoldenGate conversion service, the procedure is quite straightforward.

In this case, your converter is a function that gathers the input and output requirements from its arguments, and executes the external UNIX command (for instance, by calling the <code>system(2)</code> function).

Your function should do as much checking as possible to ensure that the external command can work. For instance, you should verify that the command is installed before calling it, and that you have execute permission.

Also check for appropriate permissions to read input files and write output files, in the case of file converters. If you detect an error before calling the command, return an error status in the status field of the data argument.

For example, the code below shows a FileToFile converter that wraps an existing UNIX command rtf2html. You will find other fully annotated examples at the end of this section.

```
#include <libgen.h>
#include <SgCvt.h>
void RtfToHtml(void *arg)
   SgCvtConverterData *data = (SgCvtConverterData *) arg;
   SgCvtContextInfo ctx_info;
   char cmd[BUFSIZ];
   int sys_status = 0;
   char *xlator_path;
   /* Evaluation - just accept for this example */
   if (data->operation == SG_CVT_REQ_EVALUATE) {
      data->status_return = SG_CVT_E_ACCEPT;
      return;
   /* Conversion */
   /* depends on `rtf2html' command being available */
  xlator_path = pathfind(getenv ("PATH"), "rtf2html", "rx");
   if (xlator_path == NULL) {
      data->status_return = SG_CVT_E_MISSING_COMMAND;
      return;
   (void) SqCvtGetContextInfo(data->context,
                SG_CVT_INFO_INPUT_FILE
```

Notice that GoldenGate passes the necessary information to a converter by reference. The SqCvtConverterData structure is the mechanism for this. It is defined as follows:

```
typedef struct {
            SgCvtRequestType operation;
            SgCvtConversionContext context;
            DMparams *output_params;
            SgCvtStatus status_return;
} SgCvtConverterData;
```

If your converter does not use an external command to translate the data, but does the conversion itself, the structure of the converter function is essentially the same.

You still use the SgCvtConverterData structure to communicate with GoldenGate. Between extracting the necessary arguments from the structure and returning from the function, you just call your own functions that do the conversion.

Input and Output Formats

Your converter should use standardized names for its input and output types wherever possible. This is important because applications are written to request data by a particular name. If your converter uses a different name for the same data format, GoldenGate will not find your converter and the conversion may fail.

See "Supported Target Formats" on page 101 for the data formats supported by the default Silicon Graphics converters.

You can also use your own data format names. However, the name your application uses must match the name you registered so GoldenGate call find the converter. However, if you use your own data format names, it is unlikely that other applications will be able to take advantage of your converter. Do this only if the format name is well understood among all the applications you intend to cooperate with.

Process Blocking

You can use SgCvtGetData() and SgCvtSendData() in either blocking or non-blocking mode, depending on your requirements. Both modes are described in "Converting Data Using File Converters" on page 328.

If your converter needs to return immediately to do other work, such as tracking activity on an I/O device, you should set the *canwait* argument to these functions to B_FALSE. If the conversion pipeline is not ready for an immediate read or write operation, the call will return immediately with a status value indicating that nothing happened and that you should try the same operation again. For additional information on the *canwait* argument, see "Sending and Receiving Data" on page 331.

If SgCvtSendData() cannot send data immediately and *canwait* is B_FALSE, it returns SG_CVT_E_AGAIN. This indicates that your data has not been sent, and you should try the operation again, using the same data.

SgCvtGetData() returns SG_CVT_E_QUEUE_EMPTY if there is no data immediately available and *canwait* is B_FALSE. You should try the operation again later.

If your converter has no other I/O requirements, you can simplify your code slightly by setting the *canwait* argument to B_TRUE. You should use this option by default, because it can eliminate redundant context switching to your idling converter, and improve system performance.

There are two categories of converter: FileToFile and StreamToStream.

A FileToFile converter uses the input and output file attributes of the conversion context to get its input and save its output, as shown in the example above.

A StreamToStream converter follows this general procedure after extracting the required parameters from the context:

- Fetch a block of input data using SgCvtGetData
- Convert the data to the new format

Send converted data back to GoldenGate

The converter repeats these steps until it receives a status of SG_CVT_E_END_OF_STREAM from SgCvtGetData, and it successfully sends all the converted data. Then it calls SgCvtSendEndOfStream to tell GoldenGate it is finished converting, and finally it returns.

The functions used for stream conversion are the same ones used by applications to work with conversion streams:

- To fetch input and output parameters to be used in the conversion, use SgCvtGetContextInfo.
- To fetch a block on data for conversion, use SgCvtGetData.
- To send a block of converted data back to GOldenGate, use SgCvtSendData.
- To break your connection to the stream and tell GoldenGate your converter is finished, use SgCvtSendEndOfStream.

Programming Constraints

Keep in mind the following constraints when writing converters:

- You must not use libraries that are unsafe for threads. For instance, you should not use Motif or other GUI libraries that are not "multi-thread-safe."
- You should be careful if installing global event handlers, such as timers and signal
 handlers, if they override those that may already be installed by the host
 application. The safest policy is to avoid this altogether.
- Where possible, you should avoid intentionally locking system resources such as
 physical memory blocks by using low-level UNIX calls or device drivers, because
 this can result in deadlock.
- Your code should be reentrant. This means it should not rely on global state between calls, because it is possible for more than one instance of your converter to be running at the same time.

Converters are free to choose the size of the data blocks they read and write. GoldenGate writes into the buffer that your converter supplies during a SgCvtGetData() call. Your converter must allocate and free this buffer space as necessary. During a SgCvtSendData() call, your converter again supplies a buffer of data. The SgCvtSendData() call does not alter your buffer. If the call returns SG_CVT_E_SUCCESS to indicate that your data has been sent, or SG_CVT_E_FAILURE

to indicate a general failure, free the buffer or re-use it as appropriate. If the call returns SG_CVT_E_AGAIN (you passed B_FALSE as the **canwait** argument) your data has not been sent, and you should retain it to try again later.

Example of a Simple Stream Converter

The following example shows a simple stream converter. It expects a stream of ASCII text characters, and outputs the stream with any uppercase characters replaced by their lowercase equivalents.

```
#include <SgCvt.h>
#include <dmedia/dm_params.h>
void CvtToLower
        (
       void *arg
   SgCvtConverterData *data = (SgCvtConverterData *) arg;
   SqCvtStatus s;
  char buf[BUFSIZ];
   size_t nreq = BUFSIZ;
  unsigned int len=0;
   int start = 0;
   int i;
   /* Evaluation */
   if (data->operation == SG_CVT_REQ_EVALUATE) {
         * In less trivial converters, we would check for
         * valid params in the context, but in this case all
         * we're doing is byte translation, so we can always
         * say yes.
         * /
        data->status_return = SG_CVT_E_ACCEPT;
        return;
   }
    * Conversion Loop. A similar construct will appear in
    * all streaming converters. The model is fetch data,
    * convert it and forward it, until we have forwarded the
```

```
* end of stream, then jump out the loop.
for (;;) {
   s = SgCvtGetData(data->context, nreq, buf, &len, NULL, B_TRUE);
   if (s == SG_CVT_E_FAILURE) {
      fprintf(stderr, "converter: failed to get data\n");
     return;
   }
   if (s == SG_CVT_E_END_OF_STREAM) {
     SgCvtSendEndOfStream(data->context);
    break;
   }
   /*** start converter-specific part ***/
   for (i=0; i<len; i++)
     buf[i] = tolower(buf[i]);
   /*** end converter-specific part ***/
   s = SgCvtSendData(data->context, (void *)buf,
                    len, NULL, B_TRUE);
  if (s == SG_CVT_E_FAILURE) {
      fprintf(stderr, "converter: failed to get data\n");
     return;
   start += (len);
}
 * When we get here, this converter's work
 * is complete. Others in the same pipeline may
 * still be running, but that's irrelevant to us.
 * We simply return. If we were invoked in a dedicated
 * sproc "thread", which is always the case for
 * streaming converters, this terminates it.
 * /
return;
```

Note the above comment about other converters: Others in the same pipeline may still be running. It is important to remember that your converter is almost always invoked as a subprocess of the application. "Programming Constraints" on page 344 lists some considerations when writing converter code.

Building a DSO

GoldenGate converters reside in Dynamic Shared Object (DSO) libraries.

After you have written and tested your conversion function by calling it directly from a test program, you are ready package it as a GoldenGate converter.

This section covers the following topics:

- "Creating a DSO For Your Converter"
- "Creating a Converter Description File"

Creating a DSO For Your Converter

Create a DSO for your converter. A simple Makefile (below) for the previous example, "Example of a Simple Stream Converter" on page 345, illustrates the compilation and linkage requirements for a GoldenGate DSO.

Creating a Converter Description File

After you compile your converter, you must create a **converter description file** that identifies your converter to GoldenGate. You use this file to test your converter, and intimately to register it with GoldenGate. A simple example for the <code>CvtToLower</code> converter follows.

```
#
# Lowercase Text Stream
#
Converter {
    Name: "CvtToLower"
    IOMethod: StreamToStream
    Input: "MIXEDCASE"
    InputLabel: "ASCII bytes, any case"
    Output: "LOWERCASE"
    OutputLabel: "ASCII bytes, lower case"
    Vendor: "SGI (Sample)"
    Version: "1.0"
    Description: "Lowercases chars in input stream"
    DSO: "/usr/people/fred/libFredsCvtrs.so"
    Function: "CvtToLower"
}
```

Make sure the DSO field is set to the full pathname for the DSO you have built.

The grammar of the converter description file is fairly simple. Three types of statements exist; they are identified by the keywords **Parameter**, **Converter**, and **Pipeline**. Table G-4 defines the statements.

Table G-4 Converter Description File Statements

Statement	Description
Parameter statement	Defines a single parameter
Converter statement	Describes a converter and may include Parameter statements
Pipeline statement	Defines a series of converters to be used together, and may contain both Converter and Parameter statements

Some example descriptions follow. The easiest way to write a converter description file is to copy an existing one. You can use these examples, or copy entries from the default registry file, /var/GoldenGate/ConverterRegistry.

```
# Lowercase Text Stream
Converter {
        Name:
                        "CvtToLower"
        IOMethod:
                        StreamToStream
        Input:
                        "MIXEDCASE"
        InputLabel:
                        "ASCII bytes, any case"
        Output:
                        "LOWERCASE"
                        "ASCII bytes, lower case"
        OutputLabel:
        Vendor:
                        "SGI (Sample)"
        Version:
                        "1.0"
        Description:
                        "Lowercases chars in input stream"
        DSO:
                        "/usr/people/fred/libFredsCvtrs.so"
        Function:
                        "CvtToLower"
# Windows BMP to XWD
Converter {
        Name:
                        "BMP_FILE_TO_XWD_FILE"
        IOMethod:
                        FileToFile
        Input:
                        "BMP_FILE"
        InputLabel:
                        "BMP_FILE"
        Output:
                        "XWD_FILE"
        OutputLabel:
                        "XWD_FILE"
        Vendor:
                        "SGI"
        Version:
                        "1.0"
        Description:
                        "BMP_FILE to XWD_FILE"
        DSO:
                        "libcvt_SGI.so"
        Function:
                        "xwdout"
# Windows BMP to Compuserv GIF-89, through JPEG (JFIF)
# This isn't necessary, since the default converters
# can go directly to GIF 89 from BMP, but it illustrates the # Pipeline
syntax for chaining converters together.
```

```
Pipeline {
        Name:
                         "BMP_FILE_TO_GIF_89_FILE"
        IOMethod:
                         FileToFile
        Input:
                         "BMP_FILE"
                         "Windows BMP"
        InputLabel:
                         "GIF_89_FILE"
        Output:
                         "Compuserve GIF"
        OutputLabel:
        Vendor:
                         "SGI"
                         "1.0"
        Version:
        Description:
                         "Windows BMP to GIF, via JPG"
   Converter {
        Name:
                         "BMP_FILE_TO_JFIF_FILE"
        IOMethod:
                         FileToFile
        Input:
                         "BMP_FILE"
        InputLabel:
                         "BMP_FILE"
                         "JFIF_FILE"
        Output:
        OutputLabel:
                         "JFIF_FILE"
        Vendor:
                         "SGI"
                         "1.0"
        Version:
        Description:
                         "BMP_FILE to JFIF_FILE"
        DSO:
                         "libcvt_SGI.so"
        Function:
                         "jfifout"
   Converter {
        Name:
                         "JFIF_FILE_TO_GIF_89_FILE"
        IOMethod:
                         FileToFile
                         "JFIF_FILE"
        Input:
        InputLabel:
                         "JFIF_FILE"
        Output:
                         "GIF_89_FILE"
        OutputLabel:
                         "GIF_89_FILE"
                         "SGI"
        Vendor:
                         "1.0"
        Version:
        Description:
                         "BMP_FILE to GIF_89_FILE"
        DSO:
                         "libcvt_SGI.so"
        Function:
                         "gifout"
```

Testing Your Converter

To test your converter, first verify that your converter description file is valid and does not cause the GoldenGate built-in registry parser to fail.

Set the environment variable CVT_REGISTRY_OVERRIDE to the full pathname of the converter description file you just created:

```
setenv CVT_REGISTRY_OVERRIDE /usr/people/fred/my_registry.cdf
```

Then run a test program that will exercise the parser. The <code>gg_query</code> demo program that comes with GoldenGate is good for this. Find it in /usr/share/src/GoldenGate (if you haven't already done so, install the demo programs from your IRIX distribution media). Copy the demo programs to your own directory, go into the <code>Query</code> subdirectory, and type <code>make</code>. Then execute the <code>gg_query</code> program. The output should look like this:

```
Converter (CvtToLower):

method: Stream To Stream
input: MIXEDCASE (ASCII bytes, any case)
output: LOWERCASE (ASCII bytes, lower case)
vendor: SGI (Sample)
version: 1.0
descr.: Lowercases chars in input stream
DSOname: /usr/people/fred/libFredsCvtrs.so
Function: CvtToLower
```

If you see an error message, go back and check that your converter description file is valid, checking especially that all string values are properly quoted. Also check that the GoldenGate software is properly installed by unsetting the CVT_REGISTRY_OVERRIDE variable and re-executing the gg_query program. It should list the default converters installed on the system (over 100 of these exist).

Once the test runs successfully, you are ready to try executing your converter. You can use your own program, or the demo programs in the ConvertFile and ConvertStream directories to do this. Each program prints a help message describing its arguments if you run it with no arguments.

After you are satisfied that your converter works when executed via GoldenGate, you are ready to make it available to other applications on the system. Unset the CVT_REGISTRY_OVERRIDE variable; you are finished unit-testing your converter.

Registering Your Converter

To register your converter, you must add your converter description file to the system registry.

The System Registry

The system registry is a text file that uses the same syntax as your converter description file. Just edit the file /var/GoldenGate/ConverterRegistry (you must be a privileged user to do this) and add your entry wherever you like.

Look at the attributes of the converters already registered. If there are potential clashes with your converter, you may wish to insert your converter closer to the beginning of the registry. Some applications may decide to convert using the first converter they find that appears to satisfy their requirements, rather than evaluating the alternatives. If you want to make sure this kind of application executes your converter rather than another one that could do that same conversion, insert your entry closer to the beginning of the file.

Some Registry Syntax Details

The most important fields are those that the service uses to locate the executable converter module: the DSO name and the Function name. The other fields are primarily for display by administration tools, and for applications to query the registry. The *Input* and *Output* fields are strings that must exactly match the format names that applications will use to search for converters. For instance, where there are naming conventions such as ICCCM target names, these should be used exactly.

See "Supported Target Formats" on page 101 for a list of standard input and output formats supported by the default converters supplied with GoldenGate.

Parameters can be one of two types: Constraint or Programmable.

Constraint parameters are used to specify constant values for a data attribute in the description file. When you see a constraint parameter, it means that this converter always sets the corresponding data attribute to the stored value, overriding its current value in the input.

Programmable parameters are used to specify parameters that are set at runtime based upon the requested input and output parameters. Programmable parameters are used to pass a runtime parameter to one of the stages of a pipeline. For example, if you have a two-stage pipeline designed to scale an SGI image to an arbitrary size, then convert it to JPEG, you want to pass one of the output parameters (the required output size) to the first stage of the converter. You do this by specifying a programmable parameter for the first stage.

Installing Your Converter

Make sure your installation copies the DSO containing your converter to the standard location for converter DSOs: /usr/lib/convert. If you install your library there, you can use a relative DSO name in your converter description file. If you install anywhere else, you must use a full path name in the registry to ensure that the service will find your converter.

See the GoldenGate Release Notes (type relnotes goldengate) for information about installation.

Some Sample Converters

This section presents annotated sample code for two different converters:

- The first example, "A Simple StreamToStream Converter UpperCase" on page 353, directly modifies data flowing through it.
- The second example, "A FileToFile Converter UNIX Man Page File to HTML File" on page 356, illustrates two techniques. First it serves as a basic template for FileToFile converters, and second it shows how you can wrap an external IRIX command to make it available as a GoldenGate converter.

A Simple StreamToStream Converter - UpperCase

This type of converter can often offer the best performance in many circumstances, because all the knowledge of the conversion operation is in the converter itself, and because it typically does not need to access the filesystem to achieve conversion. It is appropriate when the data format is naturally streamable, such as ASCII text or other self-identifying or raw data.

The converter used in this example performs a simple mapping of mixed-case text to uppercase text. The converter-specific parts are clearly marked. These are the lines that you will replace with your own task-specific conversion code. The remainder is boilerplate code that can be re-used in many different converters.

```
/* converter function */
void UpperCase
(
    void *arg
)
{
    SgCvtConverterData *data = (SgCvtConverterData *) arg;
    SqCvtStatus s;
```

The next 2 lines are somewhat task-specific. Your converter should use a buffer size appropriate to the data type and the task. Careful selection of a buffer size will yield better performance in many cases. For instance, if your converters needs to operate on audio or movie "frames," then you may choose to read and write buffers that represent whole numbers of frames.

```
char buf[BUFSIZ];
size_t nreq = BUFSIZ;
unsigned int len=0;
int start = 0;
int i;
```

The next part is the Evaluation section. Our converter operates on a byte stream: if a byte represents a lower-case character in the current locale, we are going to uppercase it. Otherwise it passes through untouched. It is appropriate for this converter to accept any stream; it does not need to evaluate parameters.

```
/* Evaluation */
if (data->operation == SG_CVT_REQ_EVALUATE) {
   data->status_return = SG_CVT_E_ACCEPT;
   return;
}
```

The following loop does the conversion, one block at a time. The loop terminates when the end of stream is detected.

These two lines show the entire task-specific code requirements for the uppercase text converter. Your converter will substitute its own conversion-specific code for these lines. The model is the same in each case; the converter generates a buffer to be sent from the buffer it has received, by applying a specific conversion algorithm.

```
for (i=0; i<len; i++)
  buf[i] = toupper(buf[i]);</pre>
```

When the new buffer has been generated, your converter sends it into the pipeline. In this example, the data was converted in place. Sometimes that is not possible, because the converted data will not fit in the original buffer. In these cases, your converter may allocate, populate, send, then free a dynamic buffer each time through the loop.

After sending all the converted data, and calling SgCvtSendEndOfStream, your converter can return. You should always set the status_return field.

```
data->status_return = SG_CVT_E_SUCCESS;
    return;
}
```

A FileToFile Converter - UNIX Man Page File to HTML File

Often, to convert data from one application into a form usable by another, you need to save the data to a file in one format, convert it using an IRIX command-line translator program, then open the new file using the application you want to use.

Applications using components that are integrated with GoldenGate can eliminate the intermediate end-user steps. The same external translator command can be packaged as a GoldenGate converter and invoked automatically on behalf of the user.

It is quite straightforward to integrate an existing IRIX command with GoldenGate. Your main task is to write a function that the service can invoke, which constructs from its parameters a command line for the translator program.

The converter described here provides GoldenGate access to a command named <code>man2htm1</code>, which converts <code>troff</code> source files for UNIX man pages into HTML files that can be viewed using a Web browser. It can be used by a CGI script that implements an online help system for remote users running Web browsers.

The command itself takes one argument: the input file name. It writes its output to stdout. The job of our converter is to make this interface look like any other GoldenGate converter.

Converter functions always require the SgCvt.h header file and always have the have the same calling convention:

```
#include <libgen.h>
#include <SgCvt.h>
void ManToHtml(void *arg)
{
```

First, cast the data argument to the right type.

```
SgCvtConverterData *data = (SgCvtConverterData *) arg;
```

Then, define some other local variables. Most of these are the same in every converter of this type that you write.

```
SgCvtStatus s;
SgCvtContextInfo ctx_info;
char cmd[BUFSIZ];
int sys_status = 0;
char *cmdpath;
```

Next, handle converter evaluation requests. This converter is very simplistic: there are no parameters, and it just ACCEPTs the request. In real converters, always provide proper evaluation of any parameters, especially if it is expensive for the application to try the conversion and fail.

```
/* Evaluation */
if (data->operation == SG_CVT_REQ_EVALUATE) {
   data->status_return = SG_CVT_E_ACCEPT;
   return;
}
```

The remaining code handles conversion requests. Note that it is never reached unless the caller requests conversion because the evaluate section has its own return statement.

The first thing to do is check that the program you are wrapping is installed and that you can execute it.

```
/* conversion */
cmdpath = pathfind(getenv ("PATH"), "man2html", "rx");
if (cmdpath == NULL) {
    data->status_return = SG_CVT_E_MISSING_COMMAND;
    return;
}
```

Then, extract the input and output filenames from the conversion context.

```
(void) SgCvtGetContextInfo(data->context,
    SG_CVT_INFO_INPUT_FILE |
    SG_CVT_INFO_OUTPUT_FILE,
    &ctx info);
```

At this point you know everything needed to construct the command you are going to execute. You use the UNIX <code>system(2)</code> call to execute the conversion, so the next step is to create the command line.

```
sprintf(cmd, "%s %s > %s 2> /dev/null", cmdpath,
    ctx_info.input_file, ctx_info.output_file);
```

Finally, execute the command, and set the return status to indicate whether it worked before returning.

```
sys_status = system(cmd);
data->status_return =
    sys_status ? SG_CVT_E_FAILURE : SG_CVT_E_SUCCESS;
return;
}
```

Note that your converter is normally run as a subprocess of the invoking application. You should not call <code>exit(2)</code> to terminate your converter; you should simply return, allowing GoldenGate and the operating system to take care of managing conversion threads.

Standard Menu Resources

This appendix provides examples of standard menu resources including:

- "Common Menu Bar Resources"
- "Standard File Menu Resources"
- "Standard Edit Menu Resources"

Common Menu Bar Resources

The following code defines the common menu bar resources.

```
! Common Menu Bar Resources
! Standard Menubar
! File Selected Edit View Tools Options Help
! <your_widget_name_goes_here>
*<file>.labelString:
                     File
*<file>.mnemonic:
*<selected>.labelString:
                     Selected
*<selected>.mnemonic:
*<edit>.labelString:
                     Edit
*<edit>.mnemonic:
*<view>.labelString:
                    View
*<view>.mnemonic:
```

*<tools>.labelString: Tools
*<tools>.mnemonic: T

*<options>.labelString: Options

*<options>.mnemonic:

*<help>.labelString: Help
*<help>.mnemonic: H

Standard File Menu Resources

The following code defines the standard file menu resources.

*<new>.labelString: New

*<new>.mnemonic: N

*<new>.acceleratorText: Ctrl+N

*<new>.accelerator: Ctrl<Key>N

*<reopen>.labelString: Reopen
*<reopen>.mnemonic: R

*<import>.labelString: Import
*<import>.mnemonic: I

*<save>.labelString: Save

*<save>.mnemonic: S

*<save>.acceleratorText: Ctrl+S

*<save>.accelerator: Ctrl<Key>S

*<saveas>.labelString: Save As...

*<saveas>.mnemonic: A

*<revert>.labelString: Revert *<revert>.mnemonic: *<pri>*<print>.labelString: Print... *<print>.mnemonic: *<print>.acceleratorText: Ctrl+P *<print>.accelerator: Ctrl<Key>P *<close>.labelString: Close *<close>.mnemonic: *<close>.acceleratorText: Ctrl+W *<close>.accelerator: Ctrl<Key>W *<exit>.labelString: Exit *<exit>.mnemonic: Х *<exit>.acceleratorText: Ctrl+Q *<exit>.accelerator: Ctrl<Key>Q

Standard Edit Menu Resources

The following code defines the standard edit menu resources.

! Standard Edit Menu Resources *<undo>.labelString: Undo *<undo>.mnemonic: *<undo>.acceleratorText: Ctrl+Z *<undo>.accelerator: Ctrl<Key>Z *<redo>.labelString: Redo *<redo>.mnemonic: R *<redo>.acceleratorText: Shift+Ctrl+Z *<redo>.accelerator: Shift<Key>Ctrl<Key>Z *<cut>.labelString: Cut *<cut>.mnemonic: *<cut>.acceleratorText: Ctrl+X *<cut>.accelerator: Ctrl<Key>X

*<copy>.labelString: Copy
*<copy>.mnemonic: C
*<copy>.acceleratorText: Ctrl+C
*<copy>.accelerator: Ctrl<Key>C

*<paste>.labelString: Paste
*<paste>.mnemonic: P
*<paste>.acceleratorText: Ctrl+V
*<paste>.accelerator: Ctrl<Key>V

*<clear>.labelString: Clear
*<clear>.mnemonic: e

*<delete>.labelString: Delete
*<delete>.mnemonic: D

*<select>.labelString: Select All
*<select>.mnemonic: A
*<select>.acceleratorText: Ctrl+A
*<select>.accelerator: Ctrl<Key>A

*<deselect>.labelString: Deselect All

*<deselect>.mnemonic: 1

*<deselect>.acceleratorText: Shift+Ctrl+A

*<deselect>.accelerator: Shift<Key>Ctrl<Key>A

**promote>.labelString: Promote
***promote>.mnemonic: m

*promote>.acceleratorText: Alt+Insert

****Alt<Key>Insert<Key>

*<colorEdit>.labelString: Color Editor...

*<colorEdit>.mnemonic: o

Index

Numbers	"single document, one primary", 66
3D graphics formats, 103	application programming schemes, 9
3D look icons, 187	application testing with schemes, 21
4Dwm window manager, 3, 59-81	applications automatically restart, 60 communication, 62
A	creating icons. See icons data exchange, 91-106, 319-358
access	exchanging data, 91-106, 319-358
help, 52	help, integrating, 289
adding	icon ID number, 165
icon to Icon Catalog, 173	implementation strategy, xxxii
AIFF data, 102	logging out, 60 re-invoke, 62
ALIAS data, 103	restarting, 60
alignment IconSmith, 194	save yourself example, 78
<alt> key, 220</alt>	states, 60
API	toolkits for integration, xxxii window manager, 62
GoldenGate, 106	window placement, 79-81
Apple data, 102	window size, 79-81
Apple Quicktime data, 103	ApplicationShell, 69
application integration	Arc widget, 46
checklist, xxxvii	ASCII, 236
overview, xxix-xxxviii, 3-4 schemes, 7-22	Ascii predefined file type, 310 atom
application models, 65-67	3D graphics, 103
"multiple document, no visible main", 67	audio, 102
"multiple document, visible main", 66	image, 102
"single document, multiple primary", 66	movie, 103

SGI, 101	circles
audio formats, 102	drawing, IconSmith, 198
AUTODESK data, 103	clicking the mouse, 155
AVID data, 103	ClickSpeed mouse setting, 155
В	Clipboard Selection ownership, 100
banners in window title, 79	Clipboard Transfer CLIPBOARD selection, 93 "Copy" actions, 100
Binary predefined file type, 311	"Cut" actions, 98
blocking processes, data exchange, 343	data exchange, 93-101 implementation, 98-101
book viewer setting on desktop, 156	loss of Clipboard Selection, 101 "Paste" actions, 100
BOOKVIEWER variable, 157	close
browser	window, 76
color schemes, 8	"Close" option, 76
launch in Help, 142	CMD ALTOPEN rule, 171, 220 example, 221
buttons	CMD DROP rule, 222
recycle, 43	CMD OPEN rule, 171, 219
recycle. See also recycle button	
	CMD PRINT rule, 226
С	Color Chooser, 30-35 callback resources, 34
C language operators, 305 callback	components, 31 example program, 242
Color Chooser, 34	interface, 33
	Makefile example, 244 user interaction, 34
caret cursor, 179	widget, 30-35
catalog	
icon, xxx	color image resize, 88
changing icon design size, 188	colors, 30-35
checklist	and hues, 30
creating icons, 163-175	and saturation, 30
	and values, 30
Child widget Grid widget, 49	and widgets, 30 desktop, 7-22

get and set, 34 HSV, 30 icon fill color, 197 icon outline, 197 icon shadow color, 197 icons, 197 programming, 34	process blocking, 343 registering, 352 stream converter example, 345 stream data, 330-338 StreamToStream example, 353 testing, 351 writing, 339-353
RGB, 30, 33 See also widgets	converting data GoldenGate, 105, 319-358
communication window and session manager, 68 compiling .ctr files, 172	converting files for printing, 235 co-primary windows, 71 delete protocol, 76
otr files, 172 programs with GoldenGate, 338 concave polygons IconSmith, 193	example, 71 "Copy" option, 93, 100 copy text. See data exchange
constants logical, 306	COST rule printing, 239 creating
context-sensitive help. See help continuous session management, 61 conventions	FTR file type, 230-234 creating help See help
style, xxvii conversion rules printing, 238 CONVERT rule printing, 235, 238	cross red and blue caret, 179 .ctr files, 172 cursor caret symbol, 179
converter registry, 106 GoldenGate, 320-322	customizing windows, 83-89 "Cut" option, 93, 98
converters data exchange, 106 description file, 347 DSO, 347	D
evaluation, 327 evaluation requests, 340 examples, 353 FileToFile example, 356 input and output formats, 342 installing, 353	data exchange, 319-358 exchange transferring of data, 91-106 data conversion GoldenGate, 105, 319-358

data exchange, 91-106, 319-358	stream converter example, 345
3D graphics, 103	stream data converters, 330-338
audio, 102	StreamToStream converter, example, 353
Clipboard Transfer, 93-101	StreamToStream example, 353
compiling, 338	target formats, 101-105
conversion context, 321	testing converters, 351
conversion context, setting up, 326	writing converters, 339-353
conversion overview, 320	data selection
converter attributes, 324	Primary Transfer, 95
converter description file, 347	debugging
converter evaluation, 321	session manager, 63
converter evaluation requests, 340	default printer, 240
converter example, 345	•
converter registry, 320-322	Default Viewer panel, 156
converter registry query, 323-325	delete
converter selection, 322	window, 76
converter, examples, 353	"Delete" option, 76
converter, FiletoFile example, 356	deleting
converter, registering, 352	path, IconSmith, 186
converter, testing, 351	"Deselect" button
converters, 106	IconSmith, 190
converters, evaluating, 327	design elements
converters, installing, 353	icons, sharing, 188
converters, writing, 339-353	desk management, 64
converting data, 322	screens, 64
display converter details, 328	virtual screens, 64
DSO, 347	
example converters, 353	Desks Overview window, xxx
file converters, 328-330	desktop
FileToFile example, 356	and fonts, 7-22
image, 102	book viewer setting, 157
input and output formats, 342	BOOKVIEWER variable, 157
installing converters, 353	CMD PRINT rule, 226
linking, 338	colors, 7-22
movie, 103	Default Viewer panel, 156
operators, 325	Desks Overview window, xxx
Primary Transfer, 92-93	display image files, 157
process blocking, 343	editor, 157
querying converter registry, 323-325	environment variables, 261-262
registering converters, 352 SGI, 101	example of setting preferred editor, 157
JOI, 101	Icon Catalog, xxx

icons, 161-175, 177-205	example program, 244
image viewer, 157	getting and setting values, 37
IMGVIEWER variable, 157	interface, 36
implementation strategies, xxxii	values, 37
integration checklist, xxxvii	dialogs
integration of windows, 69	creating, 73,74
look and feel, 3-4	delete example, 77
look and feel, basic steps, 4	"Delete" option, 77
mail reader, 157	example, 74
MAILBOXPROG, 157	notification, 74
menu, MENUCMD rule, 226	windows, 74
PostScript viewer, 157	DIB data, 102
printing, 226, 235-240	dimensional look
PSVIEWER variable, 157	3D and IconSmith, 187
schemes, 7-22	dircontains function, 217
setting a book viewer, 156	
setting a mailer, 156	directories
setting a PostScript viewer, 156	detecting changes, FAM, 112
setting a text editor, 156	monitoring changes, 107-123
setting a Web browser, 156	Directory predefined file type, 310
setting an image viewer, 156	drag and drop icons, 222
toolkits, xxxii	Drop Pocket, 43
turn on look and feel, 5	widget, 41-42
utilities and user preferences, 157	drop pocket
Utilities panel, 156	example program, 247
web browser, 157	Makefile example, 248
WEBBROWSER variable, 157	DROPIF rule, 223
widgets, 23-52	DSO
window editor, 157	GoldenGate converters, 347
WINEDITOR variable, 157	dtUtilities
desktop environment	variables and values, 157
introduction, xxix-xxxviii, 3-4	
Developer Magic RapidApp, xxxiv-xxxvi	dtUtilities panel, 156
device files	Dynamenu
FTRs, 217	example program, 251
Dial widget, 36-38	
appearance, 36	F
callback function, 37	E
characteristics, 36	Editmoni
detecting changes in value, 38	Edit menu
	"Copy" option, 93

"Cut" option, 93 "Paste" option, 93 edit menu resources, 361-362	suspend monitoring, 111 symbolic links, 114 theory of operation, 108 using FAM, 118-123
Editor Utilities panel, 156	waiting for changes, 118
enlarging icons, 187	file menu resources, 360-361
environment variable IMGVIEWER, 221 environment variables	File Selection Box example program, 255 Makefile example, 257 File Selection Box widget, 25
desktop, 261-262 example programs help, 265-304 widgets, 241-259	File Selection Box widget, 25 file type definition, 168 example, 171 icons, 166
exchanging data, 319-358 See also data exchange target formats, 101-105	file types predefined, FTRs, 217
Executable predefined file type, 312 "Exit" option, 75	File Typing Rules, 207-234 Also see FTRs description of, 209
explicit session management, 61 F	files converting to print, 235 creating file type, FTR, 230-234 detecting changes, FAM, 112
FAM, 107-123 See also File Alteration Monitor File Alteration Monitor, 107-123 cancel monitoring, 111 closing a connection, 109 detecting changes, 112 examples, 116-118 fam daemon, 107 include files, 108 interface, 109-118	device files, 217 .ftr, 210 icons, 162 monitoring changes, 107-123 predefined file types naming conventions, 309 printing, 236 printing from Desktop, 236 selecting, 25 types, predefined, 309-315 types, predefined. See also predefined file types
libraries, 109-118 libraries, 108 monitoring a file, 110 opening a connection, 109 polling for changes, 122 resume monitoring, 111	fill color icons, 197 FILTER rule printing, 239 Finder widget, 42-46

callback, 45	file type, creating, 230-234
components, 43	file types, predefined, 217
Drop Pocket, 43	ICON rule, 209, 227
example program, 250	icon, file type, 166
interface, 44	icons, 162, 219
path navigation, 43	LEGEND rule, 209, 218
recycle button, 43	MAP rule, 209, 225
user interaction, 45	MATCH expressions, 215-216
values, 44	MATCH rule, 209, 212
FITS data, 102	matching files, 214
fonts	matching non-plain files, 217
and desktop, 7-22	matching tagged files, 212
format	MENUCMD rule, 209, 226
files for printing, 235	naming file type, 167
formats	naming files, 167
3D graphics, 103	predefined file types, 217
audio, 102	set variables, 219
converting data, 105, 319-358	SETVAR rule, 209, 219
image, 102	SPECIALFILE rule, 209, 217
movie, 103	SUPERTYPE rule, 209, 210
SGI, 101	TYPE rule, 209, 210
ftr files, 210	writing MATCH expressions, 214
	functions
FTRs, 207-234	icon descriptive functions, 307
adding, 168 CMD ALTOPEN rule, 209, 220	
CMD DROP rule, 209, 222	•
CMD OPEN rule, 209, 219	G
CMD OPEN rule, Makefile, 220	generic icone 106
CMD PRINT rule, 209, 226	generic icons, 196
compiling source files, 172	GenericWindowExecutable predefined file type, 313
CONVERT rule, printing, 235	-geometry option, 79
creating a file type, 230-234	GIF data, 102
device files, 217	gizmos. See widgets
dircontains function, 217	GoldenGate, 319-358
directory location, 167	API, 106
DROPIF rule, 209, 223	compiling programs, 338
example, creating file type, 230-234	conversion overview, 320
file directories, 317	converter description file, 347
file type definition, 168	converter header filedata exchange
file type example, 171	API, 106

converter registry, 106, 320-322 data conversion, 105, 319-358 DSO, 347 example converters, 353 FileToFile example, 356 linking programs, 338 registering converter, 352 See also data exchange stream converter example, 345 StreamToStream example, 353	displaying help index, 132 document files, 129 DTD, 145 DTD elements, 274 errors, 149 examples, 146 files, 138-144 help button, 135 help document example, 265 helpmap file conventions, 139
testing converter, 351	helpmap file format, 139 helpmap files, 129, 138-144
Graph widget, 46-47	helpmap URL, 142
gravity controlling in IconSmith, 195	implementation, 133-138
gravity snap	include file, 128
IconSmith, 194	initializing help, 130 installation
grid snap	custom, 152
IconSmith, 194	installing, 150-152
Grid widget, 48-52	integrating in application, 289
characteristics, 48	interface, 129-133
child, 49	launch browser, 142
examples, 50	library, 128
layout, 48	Makefile, 147
grids controlling in IconSmith, 195	messages, 149 online examples, 265-304
controlling in iconomium, 173	overview, 126
	packaging, 150-152
H	references, 152
	SGML references, 152
help, 125-153	spec file, 147
application integration, 289	subsystem, 150-152
building, 147, 149	URL access, 142
compiling, 147, 149	valid elements in Help DTD, 274
constructing the menu, 134	viewer, 126-128
context-sensitive, 136	Web access, 52, 142
creating files, 146 creating installable subsystem, 151	widget hierarchies, 142 widgets and helpmap, 142
custom installation, 152	windows, 126
debugging, 149	writing, 145-150
displaying a help topic, 131	writing overview, 145
	,

writing samples, 145	custom, 162
History Button	design conventions, 196
example program, 251	directory, 178
HSV colors, 30	double-clicking, 220
HTML	drag and drop, 222
attributes, 55	drawing, 165
libraries, 52, 57	files, 162, 178
tags, 55	files type, 162
viewer component, 52, 57	fill color, 197
viewer components, 54	.fti file, 178
widget, 52, 57	FTR file type example, 171
_	functions, 307
HTML data, 105	generic, 162, 178
hypertext data, 105	generic components, 196
	icon description language, 305-308
	ICON rule, 227
I	icon status variables, 306
ICCCM 1 20F	identification number, 165
ICCCM targets, 225	installing in Icon Catalog, 173
Icon Catalog	isometric circles, 201
adding icons, 173	logical constants, 306
Icon Catalog window, xxx	Magic Carpet, 196
ICON rule, 171, 227	naming file type, 167
iconbookedit command, 173	naming FTR files, 167
icons, 161-175, 177-205	of windows, 83-89
3D look, 187	outline color, 197
adding FTRs, 168	ovals, 200
and FTRs, 162	programming, 166
badge, 165, 178	See also desktop
binary executables, 162	See also IconSmith
C language operators, 305	segments or lines, 180
caret symbol, 179	shadow color, 197
checklist, 163-175	size, 197
circles, 198	steps to creating, 163-175
CMD OPEN rule, 219	IconSmith, 177-205
compiling source files, 172	3D look, 187
composition conventions, 196	advanced techniques, 198
conditional operators, 306	align objects, 194
creating, 161-175, 177-205	caret, 179
creating file type, 166	changing design size, 188
croming the type, 100	circles, 198

colors, 197	Transformation buttons, 191
composition conventions, 196	Transformation example, 192
concave polygons, 193	transformation pin, 179
deleting path or vertex, 186	"Undo" option, 184
"Deselect" button, 190	vertex, 180
design conventions, 196	windows, 182-184
drawing filled shapes, 185	identification number
drawing paths, 185	application icons, 165
drawing tools, 183-189	IGES data, 103
enlarge icon, 191	
generic components, 196	image
gravity, 195	resize, 88
gravity snap, 194	scale, 88
grid snap, 194	image formats, 102
grids, 195	image viewer
icon description language, 305-308	setting on desktop, 156
icon size, 197	ImageFile predefined file type, 312
"Import" button, 188	IMGVIEWER environment variable, 221
invoking, 180	IMGVIEWER variable, 157
isometric circles, 201	imgworks tool, 88
menus, 181-182	<u>c</u>
ovals, 200	implementation
"Partial" button, 190	checklist, xxxii
paths, 180	schemes, 9
polygons, 184, 193	tasks, xxxii
previewing icons, 188	importing
rotate icon, 191	icon design elements, 188
scaling, 187	information
"Select All" option, 191	references, xxvi
"Select Next" option, 191	installing
selecting an object, 189	converters, 353
shapes, 185	icon in Icon Catalog, 173
sharing design elements, 188	integration
"Shear Y" transformation, 193	application, xxix-xxxviii, 3-4
shrink icon, 191	checklist, xxxvii
slider, 188	desktop, 69
starting, 180	Motif, xxxiii
stretch icon, 191	schemes, 7-22
techniques, 198	strategies and toolkits, xxxii
templates, 189	interapplication data exchange, 91-106, 319-358
tools for drawing, 183-189	incruppieudon dada exchange, 71 100,017 000

GoldenGate conversion, 105, 319-358 See also data exchange target formats, 101-105 introduction desktop, xxix-xxxviii INVENTOR data, 103 IRIS GL, xxxvii IRIS ViewKit, xxxiv-xxxvi isometric circles drawing, IconSmith, 201	linking programs with GoldenGate, 338 logical constants icon description language, 306 look and feel basic steps, 4 introduction, xxix-xxxviii overview, 3-4 schemes, 7-22 turn on, 5
J	М
JIFF data, 102 JPEG data, 102	Magic Carpet icons, 196 mail reader setting, 157 MAILBOXPROG variable, 157
κ	mailer setting on desktop, 156
keys <alt>, 220 knob widget. See Dial widget Kodak data, 102</alt>	main windows, 69 example, 69 Makefile CMD OPEN rule, 220 MAP rule, 225
L	MATCH rule, 171, 212 matching tagged files, 212 menu bar
labels minimized window, 89 LaunchExecutable predefined file type, 313 LED widget example program, 259 LEGEND rule, 171, 218 libraries GoldenGate DSO, 347 licensing on the Web, 52 line segments, 180	resources, 359-360 MENUCMD rule, 226 menus "Close" option, 76 "Delete" option, 76 edit menu resources, 361-362 "Exit" option, 75 file menu resources, 360-361 IconSmith, 181-182 menu bar resources, 359-360 standard resources, 359-362

Microsoft data, 102	See also help
middle mouse button	See help
inserting text, 97	Web access, 52
MIME types, 225	OpenGL, xxxvii
minimized windows, 83	reference, xxvi
creating, 85	operators
examples, 84	conditional, 306
using imgworks, 88	icon description routines, 305
mixed-model programming widgets, 29	OSF/Motif, 3
monitor file changes. See File Alteration Monitor	desktop integration, xxxiii
Motif	reference, xxvi
desktop integration, xxxiii	.otr files, 172
Motif window manager, 59	outline color
mouse	icons, 197
setting the click speed, 155	ovals
movie formats, 103	drawing, IconSmith, 200
MPEG data, 102, 103	overview
multiClickTime resource, 155	of desktop, xxix-xxxviii, 3-4
"multiple document, no visible main"	ownership Clipboard Selection, 100
application model, 67	1
"multiple document, visible main"	
application model, 66	Р
	pages
N	icon catalog, xxx
	"Partial" button
naming conventions	IconSmith, 190
predefined file types, 309	"Paste" option, 93, 100
navigation	path navigation, 43
path, 43	paths
NEXT data, 102	drawing, IconSmith, 185
notification dialogs, 74	segments, 180
	Percent Done Indicator
•	example program, 258
0	percent done indicator, 27
online help	persistent selection
examples, 265-304	See Primary Transfer

PHOTO data, 102	Primary Selection
PIXAR data, 104	inserting, 97
pixmap data, 102	See also Primary Transfer
placement	Primary Transfer
of windows, 79	callback function, 96
point	data exchange, 92-93
vertex, 180	data selection, 95
pointer widget. See Dial widget	implementation, 95-97 loss of, 97
polling	requests for, 96
for file changes, 122	primary windows, 69
polygons	example, 69
IconSmith, 184, 193	Print Manager
pop-up windows, 71	default printer, 240
PostScript files	printing, 235-240
printing, 236	ASCII files, 236
PostScript viewer	CONVERT rule, 238
setting on desktop, 156	converting files, 235, 236-238
PPM data, 102	COST rule, 239
predefined file types, 309-315	current printer, 240
Ascii, 310	default printer, 240
Binary, 311	FILTER rule, 239
Directory, 310	PostScript files, 236
Executable, 312	print conversion rules, 238 RGB files, 236
GenericWindowExecutable, 313	routeprint command, 235
ImageFile, 312	routing a job, 235
LaunchExecutable, 313	process blocking
naming conventions, 309	data exchange, 343
Script, 313 SourceFile, 311	programming
SpecialFile, 310	mixed model widgets, 29
ttyExecutable, 314	programs
ttyLaunchExecutable, 314	widget examples, 241-259
ttyLaunchOutExecutable, 315	progress indicator, 27
ttyOutExecutable, 314	protocols
predefined file types, FTRs, 217	close, 76
predesigned	delete, 76
icon templates, 189	delete, example, 76
preview icons, 188	quit, 75

quit, example, 75 saving state information, 77	S
window manager, 75-79	saving state information, 77
PSVIEWER variable, 157	scale image, 88
Q	Scale widget, 27 example program, 258
quit window, 75	scaling icons, 187 transformation pin, 179 schemes, 7-22
R	application testing, 21 basic concepts, 8
RapidApp, xxxiv-xxxvi	browser, 21 Color Schemes option, 21
recycle button, 43	considerations, 10
customize, 44	creating new, 21
references	customization, 8
list, xxvi	defaults, 7
removing	designing new, 21
path, IconSmith, 186	direct access, 13
resize	ease of use, 8 implementation, 9
image, 88 windows, 79	new, 21
,	non-default colors, 11
resources menus, 359-362	override the default, 11
schemes, 15	overview, 7
restarting applications, 77	pre-defined resources, 15
session management, 60	programming, 9
states, 60	schemebr browser, 8, 21 symbolic values, 15
RGB colors, 30, 33	turn on, 9
RGB files	user customization, 8
printing, 236	why use, 8
RGB image	Script predefined file type, 313
resize, 88	SD2 data, 102
rotation	segments
transformation pin, 179	IconSmith, 180
routeprint command, 235	"Select All" option
	IconSmith, 191

"Select Next" option	sliders
IconSmith, 191	color, 33
selecting	SOFTIMAGE data, 104
IconSmith, 189	SourceFile predefined file type, 311
session management, 60-64	SpecialFile predefined file type, 310
communication with window manager, 68	SPECIALFILE rule, 217
continuous, 61, 77	speed of mouse clicking, 155
debugging, 63	Springbox widget, 47-48
example, 63 explicit, 61	states
overview, 60	saving, 62
saving state information, 62,77	stream converter example, 345
WM_COMMAND, 62	stream data converters, 330-338
session manager	style
and window manager, 68	conventions, xxvii
SETVAR rule, 219	SUPERTYPE rule, 171, 210
SGI audio data, 102	support windows, 73
SGI data, 103	"Delete" option, 77
SGI help	example, 73
See help	symbolic links
sgiMode resource, 5	File Alteration Monitor, 114
shadow color	symbolic values
icons, 197	schemes, 15
shapes	syntax
filled, IconSmith, 185	conventions, xxvii
"Shear Y" transformation, 193	system
"single document, multiple primary"	default printer, 240
application model, 66	
"single document, one primary" application model, 66	Т
size, 79-81	tag command, 212
image, 88	target formats
of windows, 79	3D graphics formats, 103
sizing	audio formats, 102
icons, 187	image formats, 102
slider	movie formats, 103
IconSmith, 188	SGI formats, 101
slider widget, 27	targets

data exchange, 101-105 ICCCM, 225	Transformation button IconSmith, 191
telldesktop command, 174	transformation pin, 179
templates	ttyExecutable predefined file type, 314
icons, 189	ttyLaunchExecutable predefined file type, 314
testing application with schemes, 21	ttyLaunchOutExecutable predefined file type, 315 ttyOutExecutable predefined file type, 314
text	TYPE rule, 171, 210
copy. See data exchange text editor example setting, 157 setting on desktop, 156	types MIME, 225 predefined file types, 309-315 predefined file types. See also predefined file types
text field widget, 28	
text widget, 28 ThumbWheel widget, 38-41	U
callback structure, 40 detecting changes, 40 example program, 252 interface, 39 interface values, 39 values, 40	"Undo" option IconSmith, 184 URL access in helpmap file, 142 user interaction
TIFF data, 103	Finder widget, 45
title	Tilidel Widget, 45
minimized window, 89	
titles	V
of windows, 79	
tools application integration, xxxii IRIS GL, xxxvii OpenGL, xxxvii RapidApp, xxxiv-xxxvi ViewKit, xxxiv-xxxvi, xxxvii	variables desktop, 261-262 icon status, 306 set, FTRs, 219 vertex, 180 ViewKit, xxxiv-xxxvi
topics additional references, xxvi	desktop integration, xxxvii HTML widget, 54
tracing design templates, 189	reference, xxvi retrieve resource, 14
transferring data, 91-106, 319-358 See Primary Transfer	views desktop, xxx virtual reality data, 104

virtual screens, 64	Finder. See also Finder widget
VRML data, 104	Graph, 46-47
	Grid, 48-52
	Grid. See also Grid widget
W	History Button example program, 251
	HTML viewer component, 52, 57
WAVE data, 102	knob, 36-38
WAVEFRONT data, 104	LED example program, 259
Web access	OpenGL, 29
HTML widget, 52-57	Percent Done Indicator example, 258
information retrieval, 52	percentage done, 27
licensing, 52	programming mixed-model, 29
online help, 52	progress indicator, 27
web browser	Scale, 27
setting on desktop, 156	Scale example program, 258
2	SgColorChooser, 30
web browser setting, 157	slider, 27
WEBBROWSER variable, 157	Springbox, 47-48
wheel widget. See ThumbWheel widget	text, 28
widgets, 23-52	text field, 28
ApplicationShell, 69	ThumbWheel, 38-41
Arc, 46	ThumbWheel example program, 252
building demos, 24	ThumbWheel. See also ThumbWheel widget
Color Chooser example program, 242	window categories, 65-67
Color Chooser Makefile example, 244	implementation, 68
ColorChooser, 30-35	window management, 59-81
ColorChooser. See also colors	communication with session manager, 68
demos, 24	debugging, 63
Dial, 36-38	window manager, 3
Dial example program, 244	and session manager, 68
Dial. See also Dial widget	protocols, 75-79
Drop Pocket, 41-42	sending messages, 75
Drop Pocket example program, 247	windows, 79-81
Drop Pocket Makefile example, 248	Also see IconSmith, windows
enhanced, 25-29	banner, 79
example programs, 241-259	co-primary, 71
File Selection Box, 25	customizing, 83-89
File Selection Box example program, 255	Desks Overview, xxx
File Selection Box Makefile example, 257	desktop, xxx
Finder, 42-46	dialogs, 73, 74
Finder example program, 250	· · · · · · · · · · · · · · · · · · ·

example, creating a co-primary, 71 example, creating a dialog, 74	X
example, creating a dialog, 74 example, creating a main primary, 69 example, creating a support, 73 example, delete protocol, 77 example, quit protocol, 75 example, save yourself protocol, 78 help, 126 Icon Catalog, xxx iconified, 83-89 iconified, changing state, 89 iconified, creating, 85 iconified, examples, 84 iconified, using imgworks, 88 integration with desktop, 69 main, 69 minimized, 83-89 creating, 85 examples, 84 using imgworks, 88	X X Window System reference, xxvi Xt help See help
minimized, changing state, 89 minimized, labeling, 89	
placement, 79-81 pop-up, 71 primary, 69 protocol, 75 save yourself example, 78 save yourself protocol, 77 sending messages, 75 session management, 77 support, 73 title, 79, 89	
WINEDITOR variable, 157	
WM_COMMAND session management, 62	
writing help See help	