XRT/gear™ for Motif

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 3.0



Software Development Productivity

260 King Street East
Toronto, Ontario, Canada M5A 1K3
(416) 594-1026
www.klgroup.com

January 1999

Part No: KLGM30

Copyright © 1996-1999 by KL Group Inc. All Rights Reserved.

KL Group, the KL Group logo, XRT, XRT/3d, XRT/care, XRT/field, XRT/gear, XRT/graph, XRT/table and XRT PDS are trademarks or registered trademarks of KL Group Inc. in Canada, United States and other countries.

Printed in Canada on recycled paper. 

A Note on XRT Product Licensing

The terms and conditions of the XRT product license are detailed in the license agreement on the next page. Please take a few minutes to read and understand the agreement. The notes below are offered to aid in reading the agreement only, and are not considered part of the agreement itself.

- XRT products (such as XRT/gear, XRT/graph) are licensed on a per toolkit (such as Motif) and per architecture (such as SPARC, RS6000, HP 700/800, OpenVMS AXP, etc.) basis.
- There are two types of XRT licenses: single and network-site.
- A single license authorizes use of an XRT product for development of end-user applications on a particular machine identified by a CPU ID. Development activities can include the following:
  - compiling and linking the XRT product library into an end-user application binary.
  - running the XRT Builder tool (if one is supplied).
  - integrating the XRT product into a GUI Builder or UIMS tool. Once an XRT product is integrated into such a tool, you may use the tool for XRT development only on machines that are licensed to use the XRT product.
- A network-site development license authorizes use of an XRT product for development of end-user applications on all similar-architecture machines on the same network and located in the same building.
- You may distribute end-user application binaries including the statically-linked XRT product library anywhere, and as many times as you like, without incurring royalty or run-time fees.

XRT FOR MOTIF BINARY LICENSE AGREEMENT

Important — Read Carefully Before Opening Software Package(s), Installing or Using Software On Your Computer

By opening a sealed software package, or installing, or using this XRT product, you indicate your acceptance of the following XRT for Motif Binary License Agreement. This is a legal agreement between you (either an individual or an entity) and KL Group Inc. If you do not accept and agree with all of the terms of this agreement, promptly destroy any downloaded or installed files, and return the unopened software package(s) and all other materials with proof of payment to your place of purchase, and your license fee will be refunded.

SOFTWARE LICENSE

Please Note: This is a license agreement and **not** an agreement for sale.

Grant of License. This License Agreement ("License") permits you to use one copy of the software you purchased (the "SOFTWARE"), on a single computer after registration of that computer's CPU identification with us. If you have purchased a network site license you may also use the SOFTWARE on computers in the same building having the same architecture as a registered computer, and connected by the same local area network, provided that the total number of such computers, on which the SOFTWARE is ever used, does not exceed the maximum number permitted by the network site license which you ordered.

If you have purchased shared library versions of the SOFTWARE, the number of users of the SOFTWARE cannot exceed the maximum number permitted by the license which you ordered.

The SOFTWARE is "in use" on a computer when it is loaded into temporary memory (i.e. RAM); when it is installed into permanent memory (e.g. hard disk, CD-ROM, or other storage device); when it is being linked into an end-user application; or when it is accessed from a software development tool such as a compiler, a linker, or a GUI builder. The SOFTWARE is not considered "in use" when installed on a network file server for the sole purpose of file distribution to other computers.

Updates. This agreement will govern the terms of any SOFTWARE updates provided to you by KL Group.

Copyright. The SOFTWARE (including any images, photographs and text incorporated into the SOFTWARE) is owned by KL Group or its suppliers and is protected by Canadian and United States copyright laws and international treaty provisions. Therefore you must treat the SOFTWARE like any other copyrighted material and not reproduce it except that you may make a backup for archival purposes. You may not copy the printed materials accompanying the SOFTWARE.

Other Restrictions. You may not rent, lease, transfer, reverse engineer, decompile, disassemble, create passwords for or translate the Software, except to the extent such foregoing restriction is expressly prohibited by applicable law.

Distribution of applications you build with the SOFTWARE. You have a royalty-free right to reproduce and distribute files in the SOFTWARE that are found in the "lib" or "lib-shared" directories and which are in a form suitable for linking into object code application files (the "XRT Library Files") provided that (a) you distribute the XRT Library Files only when linked into your software product in such a way as to prohibit further relinking; (b) your software product is not an application development tool that provides similar functionality to the SOFTWARE licensed hereunder; (c) you do not use KL Group's name, logo or trademark to market your software product; (d) you include a valid copyright notice on your software product; and (e) you agree to indemnify, hold harmless, and defend KL Group and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software product. Unless otherwise specified, you may not in any manner distribute any header files or documentation (including API documentation) provided as part of the SOFTWARE.

U.S. Government Restricted Rights. The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. The SOFTWARE (including documentation) licensed in this agreement is "Commercial computer software" as defined in Federal Acquisition Regulations (FAR) Part 12 and Defense FAR Supplement (DFARS) 252.227-7014 (JUN 1995). In accordance with FAR 12.212 and DFARS 252.227.7014, notwithstanding any provision to the contrary, use, duplication and disclosure of this software is governed by the terms of this agreement. Manufacturer is KL Group Inc., 260 King Street East, Toronto, Ontario, Canada, M5A 1K3.

LIMITED WARRANTY

Limited Warranty. KL Group warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of sixty (60) days from the date of receipt. Some states/jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. This warranty may not be assigned. KL Group does not exclude product liability or liability for damages resulting out of intent or gross negligence on the part of KL Group or its leading personnel.

Customer Remedies. KL Group's and its suppliers' entire liability and your exclusive remedy shall be, at KL Group's option, either (a) return of the price paid, or (b) repair or replacement of SOFTWARE that does not meet KL Group's Limited Warranty and which is returned to KL Group with a copy of your receipt. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

Customer Remedies for Infringement. Should the SOFTWARE become or in KL Group's opinion be likely to become the subject of a claim for infringement of patent, trade secret or copyright, KL Group may (a) procure for you the right to continue to use the SOFTWARE; or (b) replace or modify the SOFTWARE to make it non-infringing, provided that the same function is performed by the replacement or modified SOFTWARE; or if (a) or (b) are not commercially reasonable, (c) terminate the license to use the SOFTWARE, remove the SOFTWARE, and grant you a credit for the license fee as depreciated on a straight-line five (5) year basis.

No Other Warranties or Obligations. To the maximum extent permitted by applicable law, KL Group and its suppliers shall have no other obligations and disclaim all other warranties, either express or implied, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose, with regard to the SOFTWARE and the accompanying printed materials. This limited warranty gives you specific legal rights. You may have others which vary from state/jurisdiction to state/jurisdiction.

No Liability for Consequential Damages. To the maximum extent permitted by applicable laws, in no event shall KL Group or its suppliers be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of information, or other pecuniary loss) arising out of the use of or inability to use this KL Group product, even if KL Group has been advised of the possibility of such damages. Because some states/jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Governing Law. If you are a resident of one of the member states of the European Union, this Agreement is governed by the laws of your country of residence, and each of the parties hereto irrevocably attorns to the jurisdiction of the courts of your country of residence. Otherwise, this Agreement is governed by the laws of the Province of Ontario, Canada. Each of the parties hereto irrevocably attorns to the jurisdiction of the courts of the Province of Ontario.

Public Domain Software. This license agreement and the warranties, remedies, limitations, conditions and restrictions contained herein do not apply to any software clearly marked as being in the public domain.

Should you have questions concerning this Agreement, or if you desire to contact KL Group for any reason, please write:

KL Group Inc., 260 King Street East, Toronto, Ontario, Canada, M5A 1K3
Phone (416) 594-1026 / Fax (416) 594-1919

Table of Contents

Preface 1
Introduction 1
New Features In XRT/gear 3.0 2
Assumptions 2
Typographical Conventions Used in this Manual 3
Related Documents 3
Technical Support 3
Product Feedback and Announcements 4

Part I: Using XRT/gear

1 Introduction to XRT/gear 7
1.1 What is XRT/gear? 7
1.2 XRT/gear and the Motif Class Hierarchy 8
1.3 Environment Variables 8
1.4 Compiling and Authorizing a Program 10
1.5 Using Resource Files and GUI Builders 11
1.6 Using C++ with XRT/gear 11
1.7 XRT/gear Internationalization 12

2 The Enhanced Label and Enhanced Pushbutton Widgets. 15
2.1 Sample Programs 15
2.2 Widget Synopses 16
2.3 Specifying the Button Type 17
2.4 Specifying a Pixmap 17
2.5 Text and Pixmap Rotation 18
2.6 Enhanced Label Border 19
2.7 Text and Pixmap Positioning 19
2.8 Specifying the Enhanced Pushbutton Arm Pixmap 20
2.9 Resource Reference 21
Enhanced Label Resources 21
Enhanced Pushbutton Resources 23

Resources Inherited by Enhanced Label 26
Resources Inherited by Enhanced Pushbutton 27
2.10 Procedures and Methods Reference 28

3 The Enhanced Toggle Widget 31
3.1 Sample Program 31
3.2 Widget Synopsis 32
3.3 Button States 32
3.4 Setting Button Colors 35
3.5 Resource Reference 35
Resources Inherited by Enhanced Toggle 39
3.6 Procedures and Methods Reference 41

4 Tabbed Dialogs 43
4.1 Introduction 43
4.2 Sample Program 44
4.3 Widget Synopses 45
4.4 Changing the Active Tab 46
4.5 Attaching Pages to Tab Buttons 47
4.6 Controlling Tab Display 48
4.7 Constraint Resources 52
4.8 Tab Button Resources 52
Specifying the Button Type 52
Tab Anchoring 53
Setting a Pixmap 53
Controlling Button Shape 54
Text and Pixmap Positioning 55
Text and Pixmap Rotation 56
Specifying Tab Button and Page Shadow Colors 57
4.9 Specifying Corner Size 57
4.10 Controlling Resizing 58
4.11 Resource Reference 59
Tab Button Resources 59
Tab Manager Resources 61
Tab Manager Constraint Resources 65
Resources Inherited by Tab Button 66
Resources Inherited by Tab Manager 68
4.12 Procedures and Methods Reference 68
4.13 Translations and Actions 71
Default Tab Button Translations 71
Tab Button Actions 71

5 The Toolbar Widget 73
5.1 Sample Program 73
5.2 Widget Synopsis 74
5.3 Specifying Orientation 74
5.4 Margin and Spacing 75
5.5 Constraint Resources 75
5.6 Adding Help Balloons 76
5.7 Resource Reference 76
Toolbar Constraint Resources 77
Resources Inherited by Toolbar 78
5.8 Procedures and Methods Reference 78

6 The Aligner Widget 81
6.1 Introduction 81
6.2 Sample Program 82
6.3 Widget Synopsis 82
6.4 Margin and Spacing 83
6.5 Aligner and XmSeparator 83
6.6 Label Orientation 84
6.7 Constraint Resources 85
6.8 Resource Reference 87
Aligner Constraint Resources 87
Resources Inherited by Aligner 89
6.9 Procedures and Methods Reference 89

7 The Outliner Widget 91
7.1 Getting Started 91
Data Organization and Display 92
Other User Interaction 92
Building a Tree Using the Outliner Widget 93
Proceeding From Here 94
7.2 Concepts 95
The Outliner Widget and Trees 95
The XrtString Object 96
XrtList Objects 96
7.3 Widget and Xt Object Synopses 97
7.4 Creating a Tree 98
Data File Format 98
Creating a Tree From a Data File 99
Creating a Tree From a Character String 101
Manually Creating a Tree 101

Preserving the Tree 102
7.5 Traversing the Tree 103
Getting the List of Node Children 104
The Current Node 104
Traversal Convenience Methods 105
7.6 Restructuring the Tree 107
Moving Nodes 107
Deleting Nodes 108
Sorting Nodes 108
7.7 Controlling Tree Display 110
Height and Width 110
Margins 112
Colors and Fonts 113
Spacing 113
Visibility 114
Text Clipping 115
Node Height 115
Focus Highlight 115
Resizing 116
Cursor Display 117
7.8 Multiple Columns 118
Defining Multiple Labels 118
Creating New Columns 120
Ordering Columns 121
Selected Columns 122
Column Labels 123
Column Indexing 123
Controlling Column Display 124
7.9 Customizing Nodes 129
Node Label 129
Node Styles 129
Icons 131
Connecting Lines and Shortcut Buttons 133
User-Defined Data 135
7.10 User Interaction 135
Selection Behavior 136
Difference Between Focus and Selected 139
Specifying Folder States 139
Retrieving the Selected Node List 140
Selection Callbacks 140
Scrollbars 141

Drag and Drop . . . 141
Activate Callbacks . . . 144
Locating Events . . . 144
Rearranging Columns . . . 145
7.11 Printing Outliner Widgets . . . 146
Generating PostScript Output . . . 146
Customizing PostScript Fonts . . . 147
Print Callbacks . . . 148
7.12 Performance Issues . . . 149
Double Buffering . . . 149
Controlling Repainting . . . 150
Fixed Column Widths . . . 150
Use Lightweight Nodes . . . 150
Make Intelligent Use of the XmNxrtGearNodeCheckForDuplicates Resource . . . 150
Do Not Create Trees with Complete Tree Hierarchy . . . 150
7.13 Resource Reference . . . 151
Outliner Resources . . . 151
XrtNodeObject Resources . . . 170
XrtNodeFolder Object Resources . . . 171
XrtNodeStyle Object Resources . . . 171
XrtColumn Object Resources . . . 173
Resources Inherited by Outliner . . . 177
Resources Inherited by the Objects . . . 177
7.14 Procedures and Methods Reference . . . 178
Outliner Procedures and Methods . . . 178
XrtNode and XrtNodeFolder Procedures and Methods . . . 185
XrtNodeStyle Procedures and Methods . . . 196
XrtColumn Procedures and Methods . . . 197
7.15 Translations and Actions . . . 198
Translations for the Node Area Clip Child . . . 198
Translations for the Column Labels Clip Child . . . 208
Translations for the XmText widget used for editing: . . . 208
Actions . . . 209
7.16 Lightweight Folder and Item Nodes . . . 212
Building a Tree Using Lightweight Nodes . . . 212
Lightweight Nodes Cast as Widgets . . . 213
Getting and Setting Lightweight Node Resources . . . 213
Destroying Lightweight Nodes . . . 214
Lightweight Folder Demo . . . 214
Lightweight Item and Folder Node Resource Reference . . . 214

Procedures and Methods Reference 215
Lightweight Folder Node Resource Reference 218
Procedures and Methods Reference 218

8 The Progress Widget 223
8.1 Sample Program 223
8.2 Widget Synopsis 225
8.3 Label and Range 225
8.4 Timeouts and TimeOut Callbacks 226
8.5 Label Display 228
8.6 Bar Display 231
8.7 Resource Reference 233
Resources Inherited by the Progress Widget 238
8.8 Procedures and Methods Reference 238

9 The Combo Box Widget 241
9.1 Sample Program 241
9.2 Widget Synopsis 242
9.3 Menu Processing 243
Specifying the Pick List 243
Menu Callbacks 245
9.4 Controlling Display Attributes 245
Arrow Display 245
Menu Alignment 248
Spacing 248
Number of Items Visible 248
9.5 User Interaction 248
9.6 Auto Matching Completion 250
9.7 Resource Reference 250
Resources Inherited by Combo Box 253
9.8 Procedures and Methods Reference 254
9.9 Translations and Actions 255
Translations 255
Actions 255

10 The PanedWindow Widget 259
10.1 Sample Program 259
10.2 Widget Synopses 261
10.3 User Interaction 261
10.4 . . . . . Resource Reference 262
PanedWindow Resources 262

PanedWindow Constraint Resources 264
Resources Inherited by PanedWindow 265
10.5 Procedures and Methods Reference 265
10.6 Translations and Actions 267
Translations 267
Actions 267

11 The Widget Tips Utility 269

11.1 Introduction 269
11.2 Enabling Widget Tips 270
11.3 Adding Widget Tips to a Widget 270
11.4 Retrieving the Widget List 271
11.5 Help Balloon Appearance 271
11.6 Display Callbacks 272
11.7 Display Delay 273
11.8 Disabling Widget Tips 274
11.9 Manually Displaying and Removing Help Text 275
11.10 Defining Help Translations 276
11.11 Resource Reference 276
11.12 Procedures and Methods Reference 278

12 The Widget Snapshot Utility 283

12.1 Introduction 283
12.2 Pixmap Snapshots 283
12.3 Writing and Printing a Pixmap 284
12.4 XImage Snapshots 285
12.5 Resource Reference 285
12.6 Procedures and Methods Reference 286

13 Lists, Strings, Colors and Icons 289

13.1 The XrtList Object 289
Sample Code 290
Creating and Destroying a List 290
Adding and Deleting List Items 290
Searching and Retrieving List Items 292
List Comparison Routines 295
Sorting a List 296
User-Defined Data 297
13.2 The XrtString Object 297
Creating and Destroying a String 297
Determining the String Type 298

Manipulating the String Value . . . 299
Comparing Two Strings . . . 299
13.3 The XrtColorEditor Object . . . 300
Sample Code . . . 300
13.4 Icons . . . 300
13.5 Procedures and Methods Reference . . . 301
XrtList Object Methods . . . 301
XrtString Object Methods . . . 310
XrtColorEditor Methods . . . 313
Icon Manipulation Methods . . . 319

Part II: Reference Appendices

A Data Types . . . 323

B Resource Types and Classes . . . 339

B.1 Aligner . . . 339
Aligner Constraint Resources . . . 339
B.2 Combo Box . . . 340
B.3 Enhanced Label . . . 340
B.4 Enhanced Pushbutton . . . 340
B.5 Enhanced Toggle . . . 341
B.6 Outliner . . . 341
Column Resources . . . 344
Node Resources . . . 344
Node Folder Resources . . . 344
Node Style Resources . . . 345
B.7 Progress . . . 346
B.8 Tab Button . . . 346
B.9 Tab Manager . . . 348
Tab Manager Constraint Resources . . . 348
B.10 Toolbar . . . 349
Toolbar Constraint Resources . . . 349
B.11 Widget Snapshot . . . 349
B.12 Widget Tips . . . 349
B.13 Paned Window . . . 350
Paned Window Constraint Resources . . . 350

C Resource File and GUI Builder Usage . . . 351

C.1 Syntax . . . 351

D UIL Arguments 353
D.1 Aligner Resources 353
D.2 Combo Box Resources 354
D.3 Enhanced Label Resources 354
D.4 Enhanced Pushbutton Resources 355
D.5 Enhanced Toggle Resources 355
D.6 Outliner Resources 355
XrtNode Resources 357
XrtFolderNode Resources 357
XrtColumn Resources 357
XrtNodeStyle Resources 359
D.7 Progress Resources 359
D.8 Tab Button Resources 360
D.9 Tab Manager Resources 360
D.10 Toolbar Resources 361
D.11 Truncated Constant Definitions 361
D.12 Paned Window Resources 362

E Sample Code 363
E.1 Examples 363
E.2 Demos 364

F Simulating the Windows 95 Look 365

Index 367

Preface

Introduction ■ *New Features In XRT/gear 3.0*
Assumptions ■ *Typographical Conventions Used in this Manual*
Related Documents ■ *Technical Support*
Product Feedback and Announcements

Introduction

Motif is composed of many different widgets such as buttons, sliders, and menus. Conceptually, XRT/gear adds a new set of widgets to Motif. This set consists of the following:

- *Enhanced Pushbutton* and *Enhanced Label*, which provide the features of the Motif PushButton and Label widgets plus additional capabilities.
- *Enhanced Toggle*, which provides the features of the standard Motif ToggleButton widget plus additional capabilities.
- *Tab Manager* and *Tab Button*, which enable you to store and display your application widgets in a sequence of file-folders.
- *Toolbar*, which is a way of containing a row of buttons or other widgets, providing an alternative to the standard Motif menu bar.
- *Aligner*, which enables you to create and align columns of label-widget pairs.
- *Outliner*, which enables you to display data organized in a hierarchical fashion.
- *Progress*, which enables you to update the end-user on the status of a task.
- *Combo Box*, which provides a convenient way to enable an end-user to choose from a list of alternatives.
- *PanedWindow*, which is a composite widget, enabling the layout of its children in a vertical or horizontal tiled format.

XRT/gear also includes the following utilities:

- *Widget Tips*, which enables you to add a pop-up help balloon to any widget in your application.
- *Widget Snapshot*, which prints the contents of any Motif widget in PostScript format, and enables you to get a pixmap image or XImage of any Motif widget.
- *Icon Library*, which is a set of 3000 public-domain icons suitable for use with any of the XRT/gear widgets. An icon viewer is included.

- *Color Editor*, which enables developers to quickly insert a smart color editor into their applications with minimum coding.

XRT/gear widgets have resources that determine how they will look and behave. Writing programs using XRT/gear is very similar to writing any other kind of Motif program; you now have more Motif widgets to work with.

New Features In XRT/gear 3.0

The following list summarizes the major new features introduced in XRT/gear 3.0, and indicates where they are discussed in this manual:

- Editability has been added to Outliner node labels . This feature is discussed in Chapter 7, which starts on page 91.
- A Color Editor object has been added to provide a way for developers to quickly provide users with a smart color editor with a minimum of coding. This object is discussed in Section 13.3, which starts on page 300.
- A PanedWindow composite widget has been added to provide a convenient way to tile widgets horizontally or vertically. This widget is discussed in Chapter 10, which starts on page 259.
- Many miscellaneous improvements have been added, including new and improved demos and examples, an auto-completion facility for combo boxes, and the implementation of numerous bug fixes and minor improvements (see the *Release and Installation Notes* document for a listing of the main bugs fixed).

Assumptions

This manual assumes that you are proficient with the C language and the Motif toolkit. C concepts such as "an array of `char *`", and "pointer to a structure" must be understood before beginning to program Motif and XRT/gear applications. An understanding of basic Motif concepts such as getting and setting resource values is required before continuing with this manual. See page 3 for information on Motif documentation.

Typographical Conventions Used in this Manual

`Typewriter Font`

- C language source code and examples of file contents.
- XRT/gear and Motif resource names and constants.
- Commands that you enter at a terminal window.

Italic Text

- Pathnames, filenames, program, procedure and parameter names.
- New terms as they are introduced, and to emphasize important words.
- Figure and table titles.
- The names of other documents referenced in this manual, such as the *Motif Programming Manual*.

Related Documents

- *Motif Programming Manual* and *Motif Reference Manual* from O'Reilly & Associates, Inc.
- *X Toolkit Intrinsics Reference Manual*, *Xlib Programming Manual* and *Xlib Reference Manual* from O'Reilly & Associates, Inc. These manuals are not required to develop applications using XRT/gear and Motif; however, an understanding of Xlib can be useful for X Window application developers.

Technical Support

KL Group provides an update and technical support package called XRT/care. Sixty days of XRT/care coverage are included from the date you purchased XRT/gear. You may purchase annual XRT/care agreements from your XRT Reseller or KL Group. XRT/care ensures you will receive all XRT/gear bug-fix releases and upgrades. It also entitles you to receive help with installation problems, programming problems or other technical issues.

If you purchased XRT/gear from KL Group, contact KL Group for technical support; if you purchased XRT/gear from an XRT reseller, please contact the reseller for technical support. The technical support contact information is on your password letter.

Many of the initial problems you may encounter are basic installation or setup problems. Consult the "Troubleshooting" section in the *Release and Installation Notes* accompanying this release for solutions to common installation and setup problems.

Any request for support **must** include your XRT/gear license serial number. Collecting the following additional information will help us serve you better:

- Identification – your name, company name, case number (if in regard to a previous case)
- Product – XRT/gear version and architecture
- Platform – the version of the operating system, X, and Motif you are using
- Problem – reproducable description of the problem you are having

Online Support Resources

KL Group's web site provides additional avenues for technical support, for example:

XRT Technical Support: (general info and links)	`http://www.klgroup.com/cs/tech/xrt/`
XRT Knowledge Base:	`http://www.klgroup.com/cgi-bin/xrt_search.cgi`
XRT Product FAQs:	`http://www.klgroup.com/cs/tech/xrt/faq/`

Product Feedback and Announcements

We are interested in hearing about how you use XRT/gear, any problems you encounter, or any additional features you would find helpful. The majority of enhancements to XRT products are the result of customer requests.

Please send your comments to:

KL Group Inc.
260 King Street East
Toronto, Ontario, M5A 1K3 Canada

Phone: (416) 594-1026
Fax: (416) 594-1919
Email: `dev_xrt@klgroup.com`
Internet: `news://news.klgroup.com/klg.forum.xrt`

While we appreciate your feedback, we cannot guarantee a response. Please do not use the `dev_xrt` email address for technical support questions.

Occasionally, we send XRT-related product announcments to our customers using an email list. To add yourself to this mailing list, send email with the word "`subscribe`" in the body of the message to `xrt_announce-request@klgroup.com`. Visit the KL Group web site at `http://www.klgroup.com` for more details.

Part I

Using XRT/gear

1

Introduction to XRT/gear

What is XRT/gear? ■ *XRT/gear and the Motif Class Hierarchy*
Environment Variables ■ *Compiling and Authorizing a Program*
Using Resource Files and GUI Builders ■ *Using C++ with XRT/gear*
XRT/gear Internationalization

1.1 What is XRT/gear?

XRT/gear is a collection of widgets and utilities that make it easier to build useful Motif applications.

XRT/gear Widgets

The following widgets are provided as part of XRT/gear:

- *Enhanced Pushbutton* and *Enhanced Label* provide the features of the standard Motif PushButton and Label widgets plus the ability to combine pixmaps with text labels. Resources control spacing, layout, rotation, borders and clipping.
- *Enhanced Toggle* provides the features of the standard Motif ToggleButton widget plus the ability to create multi-state toggles using built-in or user-defined pixmaps. A pixmap can be automatically stippled to display an insensitive state.
- *Tab Manager* and *Tab Button* enable you to store and display your application widgets in a sequence of file-folders. Resources allow you to control the location, orientation, style, spacing and rotation of tabs.
- *Toolbar* is a way of containing a row of buttons or other widgets, providing an alternative to the standard Motif menu bar: instead of hunting for a menu item, users can click on an icon. Resources control spacing, margin and orientation. Any Primitive Widget can be displayed in the toolbar.
- *Aligner* enables you to create and align columns of label-widget pairs. This makes it easy to create attractive interfaces. Resources control margins, spacing and orientation.
- *Outliner* enables you to display data organized in a hierarchical fashion. This make it easy to view the relationships between given items.

- *Progress* enables you to update the end-user on the status of a task.
- *Combo Box* provides a convenient way to enable end-users to choose from a list of alternatives.
- *PanedWindow* enables you to tile widgets horizontally or vertically.

XRT/gear Utilities

XRT/gear includes the following utilities:

- *Widget Tips* enables you to add a pop-up help balloon to any widget in your application. Your application can specify when to enable the help balloon. Resources control colors, alignment, font, delay time and widget grouping.
- *Widget Snapshot* prints the images of any Motif widget (including Manager widgets) in PostScript format. Print margins, orientation, paper size and scaling can be specified using parameters. Additional routines enable you to obtain a pixmap image or XImage of any Motif widget.
- *Icon Library* is a set of 3000 public-domain icons suitable for use with any of the XRT/gear widgets. The icons are organized by function, size and color; most are available in several sizes. An icon viewer is included; it is located in the *$XRTHOME/src/gear/demos/iconviewer* directory.
- *Color Editor* enables you to add a smart color editor to your applications with a minimum of coding. The editor allows the user to select a color by name, hexidecimal string, RGB value or HSB (Hue, Saturation and Brightness) value. The object takes care of all user interaction, as well as allocating and previewing any selected colors.

1.2 XRT/gear and the Motif Class Hierarchy

Conceptually, XRT/gear adds new widgets to the Motif toolkit. A Motif programmer manipulates these widgets using the familiar Xt Intrinsics calls. Figure 1 shows where the XRT widgets fit into the Motif class hierarchy.

The Xt Intrinsics defines an object-oriented architecture that organizes widgets into classes. For example, the Motif XmPushButton widget class is a subclass of XmLabel, which is a subclass of XmPrimitive, which is subclassed from the Xt Core widget class. Each XRT/gear widget allows you to use resources defined by its parent widget class. For example, the Enhanced Toggle widget can use resources defined for the XmToggleButton, XmLabel, XmPrimitive, and Core widget classes.

1.3 Environment Variables

Before getting started, be sure your Unix environment is set up correctly. An environment variable called *XRTHOME* should be defined as the name of the directory containing the XRT/gear distribution files.

You can determine the current value for *XRTHOME* by entering:

```
echo $XRTHOME
```

If your *XRTHOME* environment variable is not set up correctly, see your system administrator or the *Release and Installation Notes*.

Before attempting to use XRT/gear, you must be able to compile Motif applications. Consult your Motif documentation to determine where the Motif library and include files are located. Make sure you can compile and run simple Motif applications before you begin to use XRT/gear.

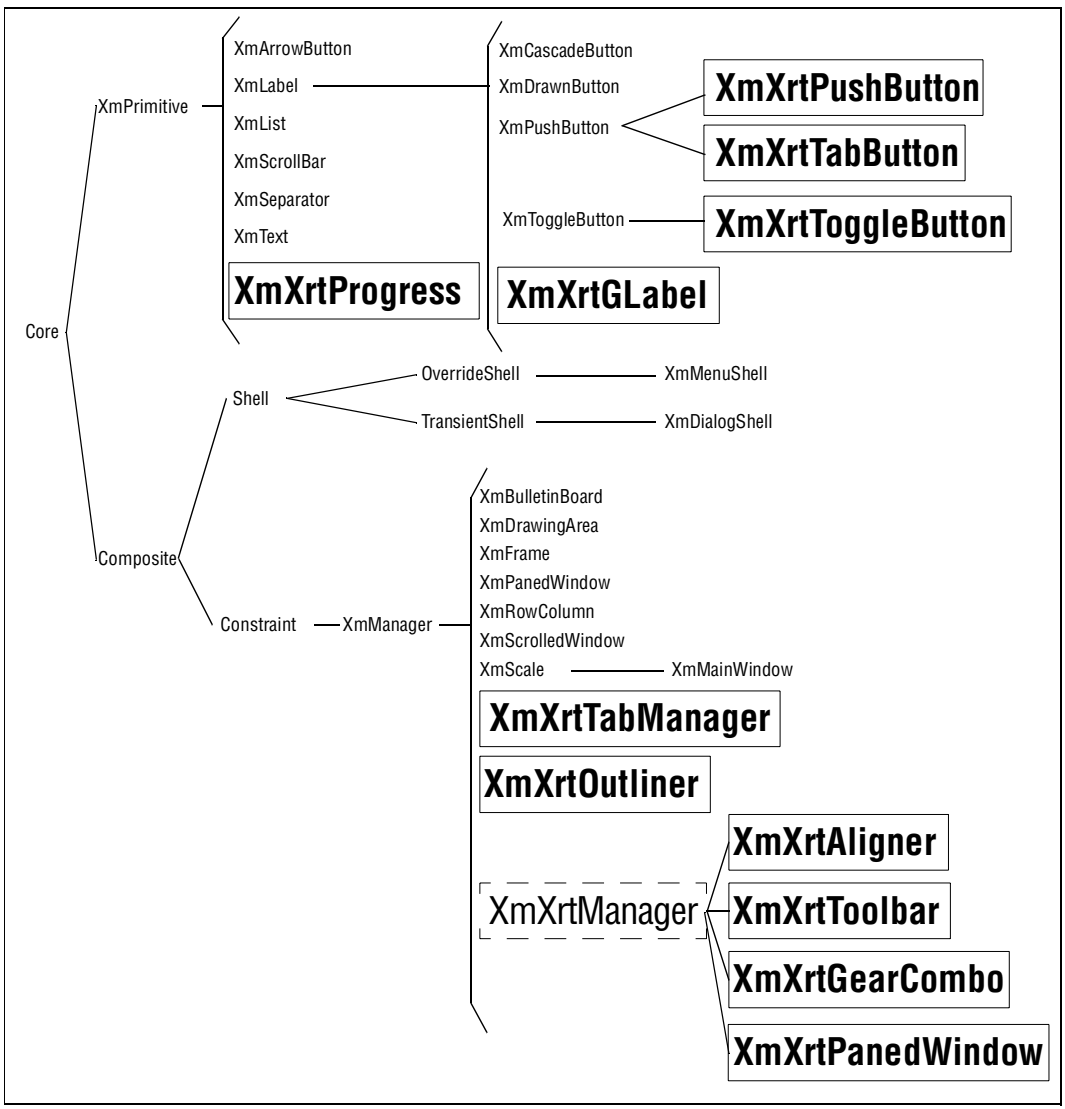


Figure 1 XRT/gear widgets' position within the Motif Class Hierarchy

1.4 Compiling and Authorizing a Program

If the X and Motif libraries and include files are installed in the standard places (normally */usr/lib*, */usr/include/X11*, and */usr/include/Xm*), a program named *simple.c* may be compiled and authorized by entering:[1]

```
cc simple.c -I$(XRTHOME)/include \
    -L$(XRTHOME)/lib -o simple \
    -lxrtgear -lXm -lXpm -lXt -lX11 -lXext -lm

$XRTHOME/bin/xrt_auth simple
```

If the libraries and include files are not installed in standard locations, additional use of the `-I` and `-L` compiler flags will be required.

xrt_auth

After the XRT/gear library has been linked into an executable application, it must be authorized with the *xrt_auth* utility. This utility will check that the machine you are building on is authorized to use XRT/gear, and will enable the application. *xrt_auth* takes two optional switches, `-v` for verbose messages and `-q` for quiet mode (that is, no messages):

```
xrt_auth [-q | -v] executable_file
```

Once an end-user application has been enabled by *xrt_auth*, it may be executed on any machine, anywhere. There are no special runtime requirements for applications built with XRT/gear.

If your application is not successfully enabled using *xrt_auth*, XRT/gear widgets will appear, but will be blank. The message shown in Figure 2 is written to *stdout*.

```
This application is built with a disabled or an expired
time-limited version of XRT/gear for Motif.  Please contact
KL Group Inc. or your XRT/gear for Motif reseller for
further information or to purchase a permanent license.
If you have a license, please check:
    1) That you have installed your license using the program
       $XRTHOME/bin/xrtpasswd or with the install script.
    2) That you have authorized your executable by running
       "xrt_auth".  (For more information refer to
       "Compiling and Authorizing a Program" in Chapter 1 of the
       XRT/gear for Motif Programmer's Guide & Reference Manual.)
    3) That you have the $XRTHOME environment variable defined.
    4) That you have permission to read the license file in
       $XRTHOME/etc/licenses.
    5) If you have a temporary license, make sure that it has
       not expired.
```

Figure 2 Message displayed by expired or non-authorized application

1. Compilation options depend on your computer architecture. See the *Makefile* in *$XRTHOME/src/gear/examples* for an example of how to compile on your system.

Note: For information on linking and authorizing shared libraries, see the note in the release notes.

C++ and ANSI C

The XRT/gear include files are compatible with C++ and ANSI C as well as with traditional "Kernighan and Ritchie" C.

1.5 Using Resource Files and GUI Builders

Sometimes it is better to use resource files instead of C code to set the value of resources. Resources files provide the following benefits:

- Speed – Changes can be seen immediately, instead of having to be compiled.
- Flexibility – You can allow end-users to modify some resource values to suit their own needs.

Using resource files, you can set almost all of XRT/gear's resources. Resource files make it easy to prototype different resource settings and quickly see the results. Appendix C on page 351 provides details on the syntax of resource settings when used in resource files.

Using GUI Builders

XRT/gear integrates with popular GUI builders. To set XRT/gear resources from a GUI builder, use the same syntax as that used in resource files. For information on this syntax, see Appendix C on page 351.

1.6 Using C++ with XRT/gear

XRT/gear provides a C++ interface to the XRT/gear widgets. To use it, include the header file *$XRTHOME/include/Xm/XrtGear.hxx* in your C++ program. This file defines classes and constructors for each XRT/gear widget, as follows:

Widget	Class	Constructor Function
Aligner	XrtAligner	XmCreateXrtAligner()
Enhanced Label	XrtGLabel	XmCreateXrtGLabel()
Enhanced Pushbutton	XrtPushButton	XmCreateXrtPushButton()
Tab Button	XrtTabButton	XmCreateXrtTabButton()
Tab Manager	XrtTabManager	XmCreateXrtTabManager()
Enhanced Toggle	XrtToggleButton	XmCreateXrtToggleButton()
Toolbar	XrtToolbar	XmCreateXrtToolbar()
Outliner	XrtOutliner	XmCreateXrtOutliner()
Progress	XrtProgress	XmCreateXrtProgress()

Widget	Class	Constructor Function
Combo Box	XrtGearCombo	XmCreateXrtGearCombo()
PanedWindow	XrtPanedWindow	XmCreateXrtPanedWindow()

The following are the classes and constructors for the objects used by the Outliner widget:

Object	Class	Constructor Function
XrtNode	XrtNode	XmCreateXrtNode()
XrtNodeFolder	XrtNodeFolder	XmCreateXrtNodeFolder()
XrtColumn	XrtColumn	XmCreateXrtColumn()
XrtNodeStyle	XrtNodeStyle	XmCreateXrtNodeStyle()

In addition to the above, the *XrtGear.hxx* file also contains the following:

- A destructor for each widget that calls *XtDestroyWidget()*.
- A method to set each settable resource. For example, the method to set `XmNxrtGearBorderType` is *setBorderType()*.
- A method to retrieve each resource. For example, the method to get the value of `XmNxrtGearBorderType` is *getBorderType()*.
- Methods to perform common Xt functions: *addCallback()*, *destroy()*, *getDisplay()*, *getName()*, *getParent()*, *isManaged()*, *isRealized()*, *manageChild()*.

C++ examples can be found in *$XRTHOME/src/gear/examples/simple_cpp.cxx* and *$XRTHOME/src/gear/examples/complex_cpp.cxx*.

1.7 XRT/gear Internationalization

ISO Latin1 Character Set

Any X Window font supported on your X Server may be used for the text in XRT/gear widgets. ISO Latin1 is the ISO standard character set for North American and West European languages. Since ISO Latin1 is a superset of ASCII, specifying a ISO Latin1 font will not change the behavior of ASCII applications. X Window fonts with names ending in *iso8859-1* are ISO Latin1 fonts.

2-byte Fonts

2-byte fonts such as Kanji may be displayed in text string, provided the type is XmString (not String).

Printing

XrtGearOutlinerVaDrawPS() and *XrtGearOutlinerDrawPS()* generate PostScript that uses the full ISO Latin1 character set if it is supported on the PostScript imaging device. 2-byte font printing is supported for Japanese characters.

2

The Enhanced Label and Enhanced Pushbutton Widgets

Sample Programs ■ *Widget Synopses*
Specifying the Button Type ■ *Specifying a Pixmap*
Text and Pixmap Rotation ■ *Enhanced Label Border*
Text and Pixmap Positioning ■ *Specifying the Enhanced Pushbutton Arm Pixmap*
Resource Reference ■ *Procedures and Methods Reference*

This chapter describes the common features of the Enhanced Label and Enhanced Pushbutton widgets, then provides information on features unique to each widget.

2.1 Sample Programs

The following code creates an Enhanced Label widget and illustrates some of Enhanced Label's capabilities:

```
#include <Xm/XrtGLabel.h>

Widget      label, form;
Pixmap      memo, memo_mask;

label = XtVaCreateManagedWidget("EnhLabel",
    xmXrtGLabelWidgetClass,         form,
    XmNlabelPixmap,                 memo,
    XmNxrtGearMask,                 memo_mask,
    XmNxrtGearLabelType,            XRTGEAR_TYPE_STRING_PIXMAP,
    XmNxrtGearStringLayout,         XRTGEAR_STRING_BOTTOM,
    XmNlabelString,                 XmStringCreateSimple("Memos"),
    NULL);
```

This creates the following widget:



Figure 3 A sample Enhanced Label

Almost all of the resources defined for Enhanced Label are also defined for Enhanced Pushbutton. The following code creates an Enhanced Pushbutton widget:

```
#include <Xm/XrtPushB.h>

Widget  button, form;
Pixmap  memo, memo_mask;

button = XtVaCreateManagedWidget("EnhButton",
    xmXrtPushButtonWidgetClass,     form,
    XmNlabelPixmap,                 memo,
    XmNxrtGearMask,                 memo_mask,
    XmNxrtGearLabelType,            XRTGEAR_TYPE_STRING_PIXMAP,
    XmNxrtGearStringLayout,         XRTGEAR_STRING_BOTTOM,
    XmNlabelString,                 XmStringCreateSimple("Memos"),
    NULL);
```

This produces the following widget:



Figure 4 A sample Enhanced Pushbutton

2.2 Widget Synopses

Enhanced Label

Include File:	*<Xm/XrtGLabel.h>*
Class Name:	XmXrtGLabel
Class Hierarchy:	Core → XmPrimitive → XmLabel → XmXrtGLabel
Class Pointer:	xmXrtGLabelWidgetClass

Enhanced Pushbutton

Include File:	*<Xm/XrtPushB.h>*
Class Name:	XmXrtPushButton
Class Hierarchy:	Core → Primitive → XmLabel → XmPushButton → XmXrtPushButton
Class Pointer:	xmXrtPushButtonWidgetClass

Since Enhanced Pushbutton is subclassed from XmPushButton, an Enhanced Pushbutton can be plugged directly into any application currently using a Motif PushButton. Similarly, an Enhanced Label can be plugged into any application currently using a Motif Label widget, except in any type of menu.

For more information on the Motif widget hierarchy and where Enhanced Label and Enhanced Pushbutton fit into this hierarchy, see section 1.2 on page 8.

2.3 Specifying the Button Type

The `XmNxrtGearLabelType` resource controls the type of label used on the Enhanced Label or Enhanced Pushbutton. This resource can be set to any of the following:

`XRTGEAR_TYPE_STRING`	label is of type XmString
`XRTGEAR_TYPE_PIXMAP`	label is a pixmap
`XRTGEAR_TYPE_STRING_PIXMAP`	label contains both an XmString and a pixmap

The default value for `XmNxrtGearLabelType` is `XRTGEAR_TYPE_STRING`.

2.4 Specifying a Pixmap

To specify a pixmap for an Enhanced Label or Enhanced Pushbutton widget, use the `XmNxrtGearLabelPixmap` resource. This resource is of type Pixmap.

This pixmap is displayed when `XmNxrtGearLabelType` is `XRTGEAR_TYPE_PIXMAP` or `XRTGEAR_TYPE_STRING_PIXMAP`. When `XmNxrtGearLabelPixmap` is not specified, `XmNlabelPixmap` is used.

`XmNxrtGearLabelPixmap` can be set in a resource file, and accepts pixmaps in XPM or XBM format.[1]

Using a Clip Mask

The `XmNxrtGearMask` resource allows you to define a clip mask for the pixmap displayed in an Enhanced Label or Enhanced Pushbutton widget. This clip mask controls what portion of the pixmap is to appear in the widget. Only non-zero values in the mask are actually drawn.

1. `XmNlabelPixmap` does not support pixmaps in XPM format.

The following code uses a clip mask to control the appearance of an image:

```
Pixmap  image, mask;
Widget  label;

label = XtVaCreateManagedWidget("EnhLabel",
    xmXrtGLabelWidgetClass,     form,
    XmNxrtGearLabelType,        XRTGEAR_TYPE_PIXMAP,
    XmNxrtGearLabelPixmap,      image,
    XmNxrtGearMask,             mask,
    NULL);
```

Figure 5 shows an example of the effects of a clip mask.



Figure 5 A clip mask and its effect

`XmNxrtGearMask` expects values of type Pixmap. The default value for this resource is `XmUNSPECIFIED_PIXMAP`. The clip mask must be of depth 1. The same file can be used to specify both a pixmap and its clip mask.

2.5 Text and Pixmap Rotation

The text and pixmap in an Enhanced Label or Enhanced Pushbutton can be rotated separately. To rotate the Enhanced Label or Enhanced Pushbutton text, set the `XmNxrtGearStringRotation` resource to one of the values shown in Figure 6.

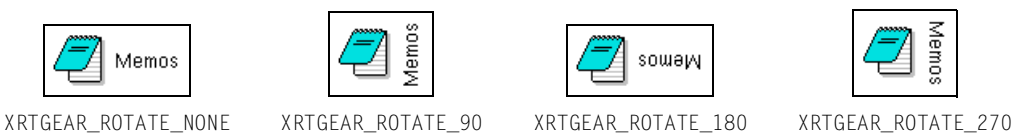


Figure 6 Rotating text

All rotations are counterclockwise. The default is `XRTGEAR_ROTATE_NONE`.

To rotate an Enhanced Label or Enhanced Pushbutton pixmap, set the `XmNxrtGearPixmapRotation` resource. This resource can be set to the same values as `XmNxrtGearStringRotation`:



Figure 7 Rotating the pixmap

The default for `XmNxrtGearPixmapRotation` is also `XRTGEAR_ROTATE_NONE`.

2.6 Enhanced Label Border

To specify the border for an Enhanced Label widget, set the `XmNxrtGearBorderType` resource. Valid values for this resource are shown in Figure 8.

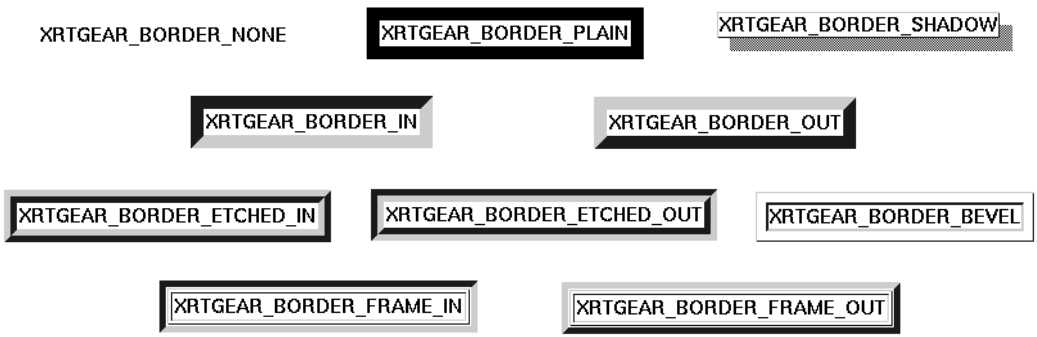


Figure 8 Enhanced Label borders

To specify the border width, set the `XmNshadowThickness` resource. By default, this resource is set to 5 by Enhanced Label (overriding the default value of 2 set by the Core resource).

`XmNxrtGearBorderType` is not defined for the Enhanced Pushbutton widget.

2.7 Text and Pixmap Positioning

When `XmNxrtGearLabelType` is set to `XRTGEAR_TYPE_STRING_PIXMAP`, both a text string and a pixmap are displayed in the Enhanced Label or Enhanced Pushbutton. Two resources control the relationship between the text string and the pixmap:

- `XmNxrtGearLayoutSpacing`, which specifies the distance in pixels between the text string and the pixmap.
- `XmNxrtGearStringLayout`, which specifies the relative position of the text string with respect to the pixmap.

XmNxrtGearStringLayout can be set to any of the values shown in Figure 9.

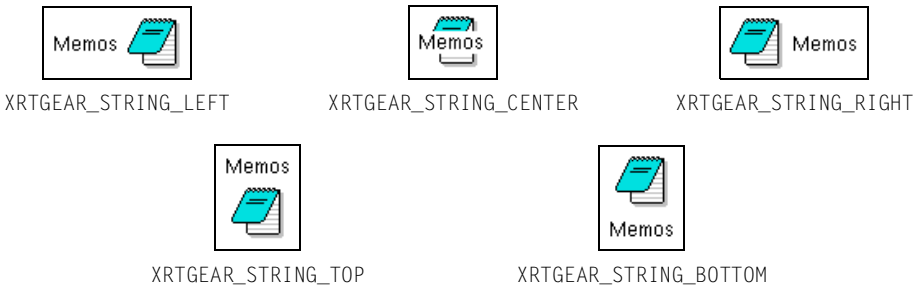


Figure 9 String and pixmap positioning

The default value for XmNxrtGearStringLayout is XRTGEAR_STRING_RIGHT.

XmNxrtGearLayoutSpacing and XmNxrtGearStringLayout are ignored unless XmNxrtGearLabelType is set to XRTGEAR_TYPE_STRING_PIXMAP.

XmNxrtGearActive controls whether the push button is drawn with the standard raised border or not. When set to False, the default, the border is drawn. When set to True, the border is not drawn, but displays when the mouse pointer is over the button.

2.8 Specifying the Enhanced Pushbutton Arm Pixmap

Enhanced Pushbutton enables you to define a pixmap to be displayed when the end-user has pressed the button but not yet released it. This pixmap is called an *arm pixmap*.

To define an arm pixmap, set the XmNxrtGearArmPixmap resource. This pixmap is displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_PIXMAP or XRTGEAR_TYPE_STRING_PIXMAP. The following is an example of an Enhanced Pushbutton with XmNxrtGearArmPixmap set:

```
button = XtVaCreateManagedWidget("EnhPushB",
    xmXrtPushButtonWidgetClass, form,
    XmNxrtGearArmPixmap,        timer,
    XmNxrtGearLabelType,        XRTGEAR_TYPE_STRING_PIXMAP,
    XmNxrtGearLabelPixmap,      smile,
    XmNlabelString,
          XmStringCreateSimple("XRT/gear's Enhanced Pushbutton"),
    NULL);
```

This produces the effect shown in Figure 10.



before and after button press during button press

Figure 10 Effect of arm pixmap

`XmNxrtGearArmPixmap` can be set in a resource file, and accepts pixmaps in XPM or XBM format.[1] (`XmNarmPixmap` does not accept XPM format.)

If `XmNxrtGearArmPixmap` is not specified, `XmNarmPixmap` is used.

To define a clip mask for the arm pixmap, set the `XmNxrtGearArmMask` resource. The clip mask must be of depth 1. The same file can be used by both `XmNxrtGearArmPixmap` and `XmNxrtGearArmMask`.

`XmNxrtGearArmPixmap` and `XmNxrtGearArmMask` are not defined for the Enhanced Label widget.

2.9 Resource Reference

This section lists all of the resources for the Enhanced Label and Enhanced Pushbutton widgets, plus some of the more commonly used inherited resources. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

2.9.1 Enhanced Label Resources

XmNbackground **Pixel** ***dynamic*** **CSG**

Specifies the background color of the label.

XmNxrtGearBorderType **XrtGearBorderType** **XRTGEAR_BORDER_NONE** **CSG**

Specifies the style used for the label border. Valid values are:

`XRTGEAR_BORDER_PLAIN`	border drawn in foreground color
`XRTGEAR_BORDER_SHADOW`	drop shadow
`XRTGEAR_BORDER_IN`	label appears inset
`XRTGEAR_BORDER_OUT`	label appears raised
`XRTGEAR_BORDER_ETCHED_IN`	double line; label appears inset
`XRTGEAR_BORDER_ETCHED_OUT`	double line; label appears raised

1. If `XmNxrtGearLabelPixmap` and `XmNxrtGearArmPixmap` are both set, the label pixmap is displayed when the button is not armed, and the arm pixmap is displayed when the button is armed. The two pixmaps are never displayed together.

XRTGEAR_BORDER_BEVEL	1-pixel in border drawn at 1/2 shadow thickness
XRTGEAR_BORDER_FRAME_IN	1-pixel shadow-in at edge; label appears framed
XRTGEAR_BORDER_FRAME_OUT	1-pixel shadow-out at edge; label appears framed
XRTGEAR_BORDER_NONE	no border

The border width is specified by XmNshadowThickness.

XmNfontList **XmFontList** ***dynamic*** **CSG**

Specifies the font list used for the widget text.

XmNforeground **Pixel** ***dynamic*** **CSG**

Specifies the foreground color of the label.

XmNheight **Dimension** ***dynamic*** **CSG**

Specifies the height of the window in pixels.

XmNxrtGearLabelPixmap **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies a label pixmap, displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_PIXMAP or XRTGEAR_TYPE_STRING_PIXMAP. XmNxrtGearLabelPixmap and XmNxrtGearMask can use the same XPM file.

XmNlabelString **XmString** ***dynamic*** **CSG**

Specifies a label string, displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_STRING or XRTGEAR_TYPE_STRING_PIXMAP.

XmNxrtGearLabelType **XrtGearType** **XRTGEAR_TYPE_STRING** **CSG**

Specifies whether a text string, pixmap, or both are displayed. Valid values are XRTGEAR_TYPE_STRING (XmString), XRTGEAR_TYPE_PIXMAP (Pixmap) or XRTGEAR_TYPE_STRING_PIXMAP. If XmNlabelType is set to XmSTRING or XmPIXMAP, it will set the corresponding XmNxrtGearLabelType resource. If both XmNlabelType and XmNxrtGearLabelType are set concurrently, XmNxrtGearLabelType will take precedence.

XmNxrtGearLayoutSpacing **Dimension** **2** **CSG**

Specifies the number of pixels between the text and pixmap. This resource is only used if XmNxrtGearLabelType is XRTGEAR_TYPE_STRING_PIXMAP.

XmNmarginHeight **Dimension** **2** **CSG**

Specifies the space between the top and bottom edges of the label and the nearest shadow edge.

XmNmarginWidth **Dimension** **2** **CSG**

Specifies the space between the left and right edges of the label and the nearest shadow edge.

XmNxrtGearMask **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies the mask to be used to clip the pixmap. Only non-zero values in the mask are drawn. The clip mask must be of depth 1. `XmNxrtGearLabelPixmap` and `XmNxrtGearMask` can use the same XPM file.

XmNxrtGearPixmapRotation **XrtGearRotation** **XRTGEAR_ROTATE_NONE** **CSG**

Specifies the rotation of the pixmap. Valid values are: `XRTGEAR_ROTATE_NONE`, `XRTGEAR_ROTATE_90`, `XRTGEAR_ROTATE_180` or `XRTGEAR_ROTATE_270`. Rotation is measured counterclockwise.

XmNshadowThickness **Dimension** **5** **CSG**

Overrides the default value (2) of this Primitive-inherited resource.

XmNxrtGearStringLayout **XrtGearStringLayout** **XRTGEAR_STRING_RIGHT** **CSG**

Specifies the relative position of the text with respect to the pixmap. Valid values are: `XRTGEAR_STRING_CENTER`, `XRTGEAR_STRING_TOP`, `XRTGEAR_STRING_LEFT`, `XRTGEAR_STRING_BOTTOM` or `XRTGEAR_STRING_RIGHT`.

This resource is ignored if `XmNxrtGearLabelType` is not `XRTGEAR_TYPE_STRING_PIXMAP`.

XmNxrtGearStringRotation **XrtGearRotation** **XRTGEAR_ROTATE_NONE** **CSG**

Specifies the rotation of the text. Valid values are: `XRTGEAR_ROTATE_NONE`, `XRTGEAR_ROTATE_90`, `XRTGEAR_ROTATE_180` or `XRTGEAR_ROTATE_270`. Rotation is measured counterclockwise.

XmNuserData **XtPointer** **NULL** **CSG**

Specifies a pointer to application-defined data.

XmNwidth **Dimension** ***dynamic*** **CSG**

Specifies the width of the window in pixels.

XmNx **Position** **0** **CSG**

The X-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

XmNy **Position** **0** **CSG**

The Y-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

2.9.2 Enhanced Pushbutton Resources

XmNxrtGearActive **boolean** **False** **CSG**

If set to True, the push button is drawn without the standard raised border, unless the mouse pointer is over the button.

XmNxrtGearArmMask **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies the clip mask to be used on the arm pixmap defined by XmNxrtGearArmPixmap. Only non-zero values in the mask are drawn. The clip mask must be of depth 1. XmNxrtGearArmPixmap and XmNxrtGearArmMask can use the same XPM file.

XmNxrtGearArmPixmap **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies a pixmap that identifies the button as armed. This pixmap is only displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_PIXMAP or XRTGEAR_TYPE_STRING_PIXMAP. XmNxrtGearArmPixmap and XmNxrtGearArmMask can use the same XPM file.

XmNbackground **Pixel** ***dynamic*** **CSG**

Specifies the background color of the button.

XmNfontList **XmFontList** ***dynamic*** **CSG**

Specifies the font list used for the widget text.

XmNforeground **Pixel** ***dynamic*** **CSG**

Specifies the foreground color of the button.

XmNheight **Dimension** ***dynamic*** **CSG**

Specifies the height of the window in pixels.

XmNxrtGearLabelPixmap **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies a label pixmap, displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_PIXMAP or XRTGEAR_TYPE_STRING_PIXMAP. XmNxrtGearLabelPixmap and XmNxrtGearMask can use the same XPM file.

XmNlabelString **XmString** ***dynamic*** **CSG**

Specifies a label string, displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_STRING or XRTGEAR_TYPE_STRING_PIXMAP.

XmNxrtGearLabelType **XrtGearType** **XRTGEAR_TYPE_STRING** **CSG**

Specifies the type of label used, and must be one of the following:

XRTGEAR_TYPE_STRING	XmNlabelString is displayed
XRTGEAR_TYPE_PIXMAP	XmNxrtGearLabelPixmap is displayed
XRTGEAR_TYPE_STRING_PIXMAP	Both XmNlabelString and XmNxrtGearLabelPixmap are displayed

If XmNlabelType is set to XmSTRING or XmPIXMAP, it will set the corresponding XmNxrtGearLabelType resource. If both XmNlabelType and XmNxrtGearLabelType are set concurrently, XmNxrtGearLabelType will take precedence.

XmNxrtGearLayoutSpacing **Dimension** **2** **CSG**

Specifies the number of pixels between the text and pixmap. This resource is only used if XmNxrtGearLabelType is XRTGEAR_TYPE_STRING_PIXMAP.

XmNmarginHeight **Dimension** **2** **CSG**

Specifies the space between the top and bottom edges of the button and the nearest shadow edge.

XmNmarginWidth **Dimension** **2** **CSG**

Specifies the space between the left and right edges of the button and the nearest shadow edge.

XmNxrtGearMask **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies the clip mask to be used on the label pixmap defined by `XmNxrtGearLabelPixmap`. Only non-zero values in the mask are drawn. The clip mask must be of depth 1. `XmNxrtGearLabelPixmap` and `XmNxrtGearMask` can use the same XPM file.

XmNxrtGearPixmapRotation **XrtGearRotation** **XRTGEAR_ROTATE_NONE** **CSG**

Specifies the rotation of the pixmap. Valid values are: `XRTGEAR_ROTATE_NONE`, `XRTGEAR_ROTATE_90`, `XRTGEAR_ROTATE_180` or `XRTGEAR_ROTATE_270`. Rotation is measured counterclockwise.

XmNxrtGearStringLayout **XrtGearStringLayout** **XRTGEAR_STRING_RIGHT** **CSG**

Indicates the relative position of the text with respect to the pixmap. Valid values are: `XRTGEAR_STRING_CENTER`, `XRTGEAR_STRING_TOP`, `XRTGEAR_STRING_LEFT`, `XRTGEAR_STRING_BOTTOM` or `XRTGEAR_STRING_RIGHT`.

This resource is ignored if `XmNxrtGearLabelType` is not `XRTGEAR_TYPE_STRING_PIXMAP`.

XmNxrtGearStringRotation **XrtGearRotation** **XRTGEAR_ROTATE_NONE** **CSG**

Specifies the rotation of the text. Valid values are: `XRTGEAR_ROTATE_NONE`, `XRTGEAR_ROTATE_90`, `XRTGEAR_ROTATE_180` or `XRTGEAR_ROTATE_270`. Rotation is measured counterclockwise.

XmNuserData **XtPointer** **NULL** **CSG**

Specifies a pointer to application-defined data.

XmNwidth **Dimension** ***dynamic*** **CSG**

Specifies the width of the window in pixels.

XmNx **Position** **0** **CSG**

The X-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

XmNy **Position** **0** **CSG**

The Y-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

2.9.3 Resources Inherited by Enhanced Label

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabel	XmNlabelString	XmLabel
XmNaccelerators	Core	XmNmappedWhenManaged	Core
XmNalignment	XmLabel	XmNmarginBottom	XmLabel
XmNancestorSensitive	Core	XmNmarginHeight	XmLabel
XmNbackground	Core	XmNmarginLeft	XmLabel
XmNbackgroundPixmap	Core	XmNmarginRight	XmLabel
XmNborderColor	Core	XmNmarginWidth	XmLabel
XmNborderPixmap	Core	XmNmarginTop	XmLabel
XmNborderWidth	Core	XmNmnemonic[a]	XmLabel
XmNbottomShadowColor	Primitive	XmNmnemonicCharSet[a]	XmLabel
XmNbottomShadowPixmap	Primitive	XmNnavigationType	Primitive
XmNcolormap	Core	XmNrecomputeSize	XmLabel
XmNdepth	Core	XmNscreen	Core
XmNdestroyCallback	Core	XmNsensitive	Core
XmNfontList	XmLabel	XmNshadowThickness	Primitive
XmNforeground	Primitive	XmNstringDirection	XmLabel
XmNheight	Core	XmNtopShadowColor	Primitive
XmNhelpCallback	Primitive	XmNtopShadowPixmap	Primitive
XmNhighlightColor	Primitive	XmNtranslations	Core
XmNhighlightOnEnter	Primitive	XmNtraversalOn	Primitive
XmNhighlightPixmap	Primitive	XmNuserData	Primitive
XmNhighlightThickness	Primitive	XmNwidth	Core
XmNinitialResourcesPersistent	Core	XmNx	Core
XmNlabelPixmap	XmLabel	XmNy	Core

a. Only supported if `XmNxrtGearStringRotation` is `XRTGEAR_ROTATE_NONE`.

2.9.4 Resources Inherited by Enhanced Pushbutton

Resource	Inherited From
XmNaccelerator	XmLabel
XmNaccelerators	Core
XmNactivateCallback	XmPushButton
XmNalignment	XmLabel
XmNancestorSensitive	Core
XmNarmCallback	XmPushButton
XmNarmColor	XmPushButton
XmNarmPixmap	XmPushButton
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	Primitive
XmNbottomShadowPixmap	Primitive
XmNcolormap	Core
XmNdefaultButtonShadowThickness	XmPushButton
XmNdepth	Core
XmNdestroyCallback	Core
XmNdisarmCallback	XmPushButton
XmNfillOnArm	XmPushButton
XmNfontList	XmLabel
XmNforeground	Primitive
XmNheight	Core
XmNhelpCallback	Primitive
XmNhighlightColor	Primitive
XmNhighlightOnEnter	Primitive
XmNhighlightPixmap	Primitive
XmNhighlightThickness	Primitive

Resource	Inherited From
XmNinitialResourcesPersistent	Core
XmNlabelPixmap	XmLabel
XmNlabelString	XmLabel
XmNmappedWhenManaged	Core
XmNmarginBottom	XmLabel
XmNmarginHeight	XmLabel
XmNmarginLeft	XmLabel
XmNmarginRight	XmLabel
XmNmarginTop	XmLabel
XmNmarginWidth	XmLabel
XmNmnemonic[a]	XmLabel
XmNmnemonicCharSet[a]	XmLabel
XmNmultiClick	XmPushButton
XmNnavigationType	Primitive
XmNrecomputeSize	XmLabel
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	Primitive
XmNshowAsDefault	XmPushButton
XmNstringDirection	XmLabel
XmNtopShadowColor	Primitive
XmNtopShadowPixmap	Primitive
XmNtranslations	Core
XmNtraversalOn	Primitive
XmNuserData	Primitive
XmNwidth	Core
XmNx	Core
XmNy	Core

a. Only supported if `XmNxrtGearStringRotation` is `XRTGEAR_ROTATE_NONE`.

2.10 Procedures and Methods Reference

This section lists the Enhanced Label and Enhanced Pushbutton procedures and methods in alphabetical order.

XmCreateXrtGLabel()

Creates an Enhanced Label widget.

```
Widget
XmCreateXrtGLabel(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmCreateXrtPushButton()

Creates an Enhanced Pushbutton widget.

```
Widget
XmCreateXrtPushButton(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmIsXrtGLabel()

Returns `True` if a widget is an Enhanced Label.

```
Boolean
XmIsXrtGLabel(
    Widget  widget
    )
```

XmIsXrtPushButton()

Returns `True` if a widget is an Enhanced Pushbutton.

```
Boolean
XmIsXrtPushButton(
    Widget  widget
    )
```

XrtGearMrmInitialize()

Registers the XRT/gear widgets with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
void
XrtGearMrmInitialize(void)
```

XrtGearSetXmStringConverter()

Sets the XmRXmString resource converter to create XmString values.

```
void
XrtGearSetXmStringConverter(void)
```

The conversion uses the following syntax to specify embedded font changes (tags):

@font@	font's tag in `XmNfontList`
@P@	previous font
\@	a literal "@"

For example,

```
For a @bold@free evaluation@P@ of any product
```

with the fontlist:

```
fixed,-*-times-bold-r-normal-*-140-*=bold
```

displays:

For a **free evaluation** of any product

3

The Enhanced Toggle Widget

Sample Program ■ *Widget Synopsis*
Button States ■ *Setting Button Colors*
Resource Reference ■ *Procedures and Methods Reference*

3.1 Sample Program

The following program creates a RowColumn widget containing three Enhanced Toggle widgets. This program illustrates the most common use of Enhanced Toggle: listing a variety of choices, one or more of which can be selected.

```
#include <stdio.h>
#include <stdlib.h>
#include <Xm/RowColumn.h>
#include <Xm/XrtToggleB.h>

int
main(int argc, char *argv[])
{
    XtAppContext                app_context;
    Widget                      toplevel;
    Widget                      menu, button1, button2, button3;

    toplevel = XtAppInitialize(&app_context, "Toggle", NULL, 0,
        &argc, argv, NULL, NULL, 0);

    menu = XmCreateRowColumn(toplevel, "laundry", NULL, 0);
    XtVaSetValues(menu,
        XmNorientation,         XmVERTICAL,
        NULL);
    button1 = XtVaCreateManagedWidget("button1",
        xmXrtToggleButtonWidgetClass, menu,
        XmNlabelString,         XmStringCreateSimple("Wash"),
        XmNxrtGearIndicator,    XRTGEAR_INDICATOR_CROSS_BOX,
        NULL);
    button2 = XtVaCreateManagedWidget("button2",
        xmXrtToggleButtonWidgetClass, menu,
        XmNlabelString,         XmStringCreateSimple("Dry"),
        XmNxrtGearIndicator,    XRTGEAR_INDICATOR_CROSS_BOX,
        NULL);
```

```
        button3 = XtVaCreateManagedWidget("button3",
            xmXrtToggleButtonWidgetClass, menu,
            XmNlabelString,         XmStringCreateSimple("Fold"),
            XmNxrtGearIndicator,    XRTGEAR_INDICATOR_CROSS_BOX,
            NULL);
        XtManageChild(menu);
        XtRealizeWidget(toplevel);
        XtAppMainLoop(app_context);
    }
```

The widgets produced by the above code are shown in Figure 11. Here, the end-user has clicked on the Wash button.



Figure 11 Sample program output

3.2 Widget Synopsis

Include File:	*<Xm/XrtToggleB.h>*
Class Name:	XmXrtToggleButton
Class Hierarchy:	Core → XmPrimitive → XmLabel → XmToggleButton → XmXrtToggleButton
Class Pointer:	xmXrtToggleButtonWidgetClass

The Enhanced Toggle widget is subclassed from the XmToggleButton widget. For more information on the Motif widget hierarchy and where Enhanced Toggle fits into this hierarchy, see section 1.2 on page 8.

Inherited Resources Not Used

Because Enhanced Toggle is subclassed from XmToggleButton, it inherits most of the features of the XmToggleButton widget. However, it does *not* use the `XmNvalueChangedCallback` and `XmNset` resources; instead, it uses the `XmNxrtGearStateChangedCallback` and `XmNxrtGearState` resources, defined later in this chapter.

3.3 Button States

By default, Enhanced Toggle widgets are either on (selected) or off (deselected). The `XmNxrtGearState` resource specifies the current state of the widget. If this resource set to `XRTGEAR_STATE_ON`, the widget is in the on state; if it is set to `XRTGEAR_STATE_OFF`, the widget is in the off state. When the end-user presses the

toggle button, this resource toggles between XRTGEAR_STATE_ON and XRTGEAR_STATE_OFF.

State Indicators

Each Enhanced Toggle widget contains an indicator which specifies the current state of the toggle button. The XmNxrtGearIndicator resource controls the style of the indicator, and is set to one of the following:

XmNxrtGearIndicator Resource Value	Resulting Indicator Style	Appearance When In Off State	Appearance When In On State
XRTGEAR_INDICATOR_NONE	indicator not displayed	indicator not visible	
XRTGEAR_INDICATOR_FILL	indicator drawn, shadows switched (default)		
XRTGEAR_INDICATOR_CHECK	checkmark drawn in the foreground color		
XRTGEAR_INDICATOR_CHECK_BOX	checkmark drawn in the foreground color and enclosed in a box		
XRTGEAR_INDICATOR_CIRCLE	circle drawn in the selected color, shadows switched		
XRTGEAR_INDICATOR_CIRCLE_BOX	circle enclosed in a box drawn in the selected color, shadows switched		
XRTGEAR_INDICATOR_CROSS	cross drawn in the foreground color		
XRTGEAR_INDICATOR_CROSS_BOX	cross drawn in the foreground color and enclosed in a box		
XRTGEAR_INDICATOR_PIXMAP	indicator is a pixmap	see below	

When XmNxrtGearIndicator is set to XRTGEAR_INDICATOR_PIXMAP, the list of indicator pixmaps to use is specified by the XmNxrtGearIndicatorPixmapList resource. This list must be terminated by NULL.[1]

1. If a pixmap in the list is XmUNSPECIFIED_PIXMAP, the last valid pixmap in the list is displayed stippled.

Clipping Indicator Pixmaps

To clip any or all of the indicator pixmaps, define a list of clip pixmaps by setting the XmNxrtGearIndicatorMaskList resource; this list also must be NULL-terminated. Each clip mask must be of depth 1, and the number of elements in this list must be the same as the number in the XmNxrtGearIndicatorPixmapList resource. If no clip pixmap is specified for a particular indicator pixmap, use UNSPECIFIED_PIXMAP for that entry in the list.

Multiple Widget States

You can specify any number of widget states you like. To do this, set the XmNxrtGearNumStates resource. When this resource is set to a value greater than 2, the value of XmNxrtGearState starts at 0 and increases by one each time the end-user presses the toggle button. When the largest state value is reached (one less than the number of states), a subsequent button press cycles XmNxrtGearState back to 0 (which is equivalent to XRTGEAR_STATE_OFF).

Multiple widget states are often used with application-defined indicator styles. The following code defines a four-state toggle button, specifying an indicator pixmap list and clip pixmap list for the four states:

```
Pixmap timerlist[5], masklist[5];

button = XtVaCreateManagedWidget("EnhToggleB",
    xmXrtToggleButtonWidgetClass,   form,
    XmNlabelString,                 XmStringCreateSimple("Time"),
    XmNxrtGearIndicatorPixmapList,  timerlist,
    XmNxrtGearIndicatorMaskList,    masklist,
    XmNxrtGearIndicator,            XRTGEAR_INDICATOR_PIXMAP,
    XmNxrtGearNumStates,            4,
    NULL);
```

State Changed Callbacks

You can define callbacks to be invoked either before or after an Enhanced Toggle widget changes state. This list of callbacks is specified by the XmNxrtGearStateChangedCallback resource.[1]

When invoked, a GearStateChanged callback is passed the following structure:

```
typedef struct {
    int     reason;     // Read-only
    XEvent  *event;     // Read-only
    int     state;
} XrtGearStateChangedCallbackStruct;
```

The *state* member specifies the new state of the toggle button. *reason* is either XRTGEAR_REASON_STATE_BEGIN (before state change) or XRTGEAR_REASON_STATE_END (after state change).

To use a GearStateChanged callback to override a state change, set *state* when *reason* is XRTGEAR_REASON_STATE_BEGIN.

1. You should not use the XmNvalueChangedCallback resource with Enhanced Toggle.

The following is a simple example of a GearStateChanged callback that counts the number of times the Enhanced Toggle has been pressed:

```
int click_count = 0;

void
stateCB(Widget w, XtPointer client_data,
    XrtGearStateChangedCallbackStruct *call_data)
{
    if (call_data->reason == XRTGEAR_REASON_STATE_BEGIN) {
        click_count++;
    }
}
```

State Change Methods

The *XrtGearToggleButtonGetState()* method gets the current state of an Enhanced Toggle widget:

```
state = XrtGearToggleButtonGetState(widget);
```

To set the current state of an Enhanced Toggle widget, call *XrtGearToggleButtonSetState()*:

```
XrtGearToggleButtonSetState(widget, XRTGEAR_STATE_OFF, False);
```

The third argument is a Boolean indicating whether the callbacks in the `XmNxrtGearStateChangedCallback` list are to be called.

3.4 Setting Button Colors

The Enhanced Toggle widget enables you to set the indicator background color. To set the indicator background when the toggle is set (when `XmNxrtGearState` is `XRTGEAR_STATE_ON`), set the `XmNxrtGearSelectColor` resource.[1] To set the indicator background when the toggle is not set (when `XmNxrtGearState` is anything other than `XRTGEAR_STATE_ON`), set the `XmNxrtGearUnselectColor` resource. Both `XmNxrtGearSelectColor` and `XmNxrtGearUnselectColor` expect values of type Pixel.

Enhanced Toggle also inherits the XmPushButton, Primitive and Core color resources; these resources are listed in the following section.

3.5 Resource Reference

This section lists all of the resources for the Enhanced Toggle widget, plus some of the more commonly used inherited resources. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

1. The `XmNxrtGearSelectColor` resource is equivalent to the `XmNselectColor` resource.

XmNarmCallback **XtCallbackList** ***dynamic*** **CSG**

Specifies a list of callbacks to be called when the button is pressed. Each routine is passed a pointer to an `XmToggleButtonCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data
        /* points to an XmToggleButtonCallbackStruct */
)
```

where `XmToggleButtonCallbackStruct` is defined as:

```
typedef struct {
    int     reason;     // Read-only
    XEvent  *event;     // Read-only
    int     set;
} XmToggleButtonCallbackStruct;
```

reason is `XmCR_ARM`. *event* points to the event structure that triggered the callback. *set* is either `True` (button is selected) or `False` (button is deselected).

XmNbackground **Pixel** ***dynamic*** **CSG**

Specifies the background color of the button.

XmNdisarmCallback **XtCallbackList** ***dynamic*** **CSG**

Specifies a list of callbacks to be called when the button is released. Each routine is passed a pointer to an `XmToggleButtonCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data
        /* points to an XmToggleButtonCallbackStruct */
)
```

where `XmToggleButtonCallbackStruct` is defined as:

```
typedef struct {
    int     reason;     // Read-only
    XEvent  *event;     // Read-only
    int     set;
} XmToggleButtonCallbackStruct;
```

reason is `XmCR_DISARM`. *event* points to the event structure that triggered the callback. *set* is either `True` (button is selected) or `False` (button is deselected).

XmNfontList **XmFontList** ***dynamic*** **CSG**

Specifies the font list used for the widget text.

XmNforeground **Pixel** ***dynamic*** **CSG**

Specifies the foreground color of the button.

XmNheight **Dimension** ***dynamic*** **CSG**

Specifies the height of the window in pixels.

XmNxrtGearIndicator **XrtGearIndicator** **XRTGEAR_INDICATOR_FILL** **CSG**

Specifies the button indicator style. Valid values are:

XRTGEAR_INDICATOR_NONE	Indicator not displayed
XRTGEAR_INDICATOR_FILL	Indicator drawn, shadows switched
XRTGEAR_INDICATOR_CHECK	Checkmark drawn in the foreground color
XRTGEAR_INDICATOR_CHECK_BOX	Checkmark drawn in the foreground color and enclosed in a box
XRTGEAR_INDICATOR_CIRCLE	Circle drawn in the selected color, shadows switched
XRTGEAR_INDICATOR_CIRCLE_BOX	Circle enclosed in a box drawn in the selected color, shadows switched
XRTGEAR_INDICATOR_CROSS	Cross drawn in the foreground color
XRTGEAR_INDICATOR_CROSS_BOX	Cross drawn in the foreground color and enclosed in a box
XRTGEAR_INDICATOR_PIXMAP	Pixmap specified by XmNxrtGearIndicatorPixmapList

If the XmNxrtGearState is XRTGEAR_STATE_INDETERMINATE or greater and XmNxrtGearIndicator is not XRTGEAR_INDICATOR_PIXMAP or XRTGEAR_INDICATOR_PIXMAP_NONE, the indicator is drawn stippled. If XmNxrtGearState is XRTGEAR_STATE_INDETERMINATE or greater and XmNxrtGearIndicator is XRTGEAR_INDICATOR_PIXMAP_NONE, the label and button are stippled with a combination of the XmNxrtGearSelectColor and the XmNxrtGearUnselectColor.

XmNxrtGearIndicatorMaskList **XrtGearIndicatorMaskList** **NULL** **CSG**

Supplies a list of clip pixmaps (clip masks) to be used with the indicator pixmap list specified by XmNxrtGearIndicatorPixmapList. Each clip pixmap controls the display of its corresponding indicator pixmap. If a given pixmap is XmUNSPECIFIED_PIXMAP, the corresponding indicator pixmap is not clipped. Each clip pixmap must be of depth 1. A clip pixmap and its corresponding indicator pixmap can be obtained from the same XPM file.

XmNxrtGearIndicatorPixmapList **XrtGearIndicatorPixmapList** **NULL** **CSG**

Supplies a list of indicator pixmaps to be used if XmNxrtGearIndicator is XRTGEAR_INDICATOR_PIXMAP. If this resource is not set, no indicator is displayed. The position of the pixmap within the list is specified by the XmNxrtGearState value. A pixmap must be supplied for each of the possible XmNxrtGearState values. If a given pixmap is XmUNSPECIFIED_PIXMAP, the last valid pixmap in the list is displayed stippled.

XmNlabelString **XmString** ***dynamic*** **CSG**

Specifies the label string.

XmNmarginHeight **Dimension** **2** **CSG**

Specifies the space between the top and bottom edges of the button and the nearest shadow edge.

XmNmarginWidth **Dimension** **2** **CSG**

Specifies the space between the left and right edges of the button and the nearest shadow edge.

XmNxrtGearNumStates **int** **2** **CSG**

Specifies the maximum number of states that can be set. The default is to allow the button to be set to `XRTGEAR_STATE_OFF` and `XRTGEAR_STATE_ON`. The value of this resource is forced to 2 if the toggle is in an XmRowColumn whose `XmNradioBehavior` is `XmONE_OF_MANY`.

XmNxrtGearSelectColor **Pixel** ***dynamic*** **CSG**

Specifies the color to set the indicator background when the toggle is set, equivalent to `XmNselectColor`.

XmNxrtGearState **int** ***dynamic*** **CSG**

Specifies the current state of the button. For convenience, the following enums are provided: `XRTGEAR_STATE_OFF`, `XRTGEAR_STATE _ON` and `XRTGEAR_STATE_INDETERMINATE`.

The number of states is specified by `XmNxrtGearNumStates`. As the end-user clicks on the button, the value of `XmNxrtGearState` increments through the states, and then back to 0 (`XRTGEAR_STATE_OFF`).

XmNxrtGearStateChangedCallback **XtCallbackList** ***dynamic*** **CSG**

Specifies a callback list to be invoked whenever the toggle button changes state. Each routine is passed a pointer to an `XrtGearStateChangedCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data
        /* points to an XrtGearStateChangedCallbackStruct */
)
```

where `XrtGearStateChangedCallbackStruct` is defined as:

```
typedef struct {
    int     reason;     // Read-only
    XEvent  *event;     // Read-only
    int     state;
} XrtGearStateChangedCallbackStruct;
```

reason is either `XRTGEAR_REASON_STATE_BEGIN` (before state change) or `XRTGEAR_REASON_STATE_END` (after state change); *event* points to the event structure that triggered the callback; *state* is the current state of the toggle button.

XmNxrtGearUnselectColor **Pixel** ***dynamic*** **CSG**

Specifies the color to set the indicator background when the toggle is not set (`XmNxrtGearState` is not `XRTGEAR_STATE_ON`). On a color terminal, the default is a color

between the background color and the top shadow color; on a monochrome terminal, the default is the background color.

XmNuserData **XtPointer** **NULL** **CSG**

Specifies a pointer to application-defined data.

XmNwidth **Dimension** ***dynamic*** **CSG**

Specifies the width of the window in pixels.

XmNx **Position** **0** **CSG**

The X-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

XmNy **Position** **0** **CSG**

The Y-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

3.5.1 Resources Inherited by Enhanced Toggle

Resource	Inherited From
XmNaccelerator	XmLabel
XmNaccelerators	Core
XmNalignment	XmLabel
XmNancestorSensitive	Core
XmNarmCallback	XmToggleButton
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	Primitive
XmNbottomShadowPixmap	Primitive
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNdisarmCallback	XmToggleButton
XmNfontList	XmLabel
XmNforeground	Primitive

Resource	Inherited From
XmNlabelString	XmLabel
XmNlabelType	XmLabel
XmNmappedWhenManaged	Core
XmNmarginBottom	XmLabel
XmNmarginHeight	XmLabel
XmNmarginLeft	XmLabel
XmNmarginRight	XmLabel
XmNmarginTop	XmLabel
XmNmarginWidth	XmLabel
XmNmnemonic	XmLabel
XmNmnemonicCharSet	XmLabel
XmNnavigationType	Primitive
XmNrecomputeSize	XmLabel
XmNscreen	Core
XmNsensitive	Core
XmNspacing	XmToggleButton
XmNstringDirection	XmLabel
XmNtopShadowColor	Primitive

Resource	Inherited From
XmNheight	Core
XmNhelpCallback	Primitive
XmNhighlightColor	Primitive
XmNhighlightOnEnter	Primitive
XmNhighlightPixmap	Primitive
XmNindicatorOn	XmToggleButton
XmNindicatorSize	XmToggleButton
XmNindicatorType	XmToggleButton
XmNinitialResourcesPersistent	Core
XmNlabelPixmap	XmLabel

Resource	Inherited From
XmNtopShadowPixmap	Primitive
XmNtranslations	Core
XmNtraversalOn	Primitive
XmNuserData	Primitive
XmNvalueChangedCallback	XmToggleButton
XmNvisibleWhenOff	XmToggleButton
XmNwidth	Core
XmNx	Core
XmNy	Core

3.6 Procedures and Methods Reference

This section lists the Enhanced Toggle procedures and methods in alphabetical order.

XmCreateXrtToggleButton()

Creates an Enhanced Toggle widget.

```
Widget
XmCreateXrtToggleButton(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmIsXrtToggleButton()

Returns `True` if a widget is an Enhanced Toggle.

```
Boolean
XmIsXrtToggleButton(
    Widget  widget
    )
```

XrtGearMrmInitialize()

Registers the XRT/gear widgets with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
void
XrtGearMrmInitialize(void)
```

XrtGearSetXmStringConverter()

Sets the XmRXmString resource converter to create XmString values.

```
void
XrtGearSetXmStringConverter(void)
```

The conversion uses the following syntax to specify embedded font changes (tags):

@font@	font's tag in `XmNfontList`
@P@	previous font
\@	a literal "@"

For example,

```
For a @bold@free evaluation@P@ of any product
```

with the fontlist:

```
fixed,-*-times-bold-r-normal-*-140-*=bold
```

displays:

For a **free evaluation** of any product

XrtGearToggleButtonGetState()

Gets the current state of an Enhanced Toggle widget.

```
int
XrtGearToggleButtonGetState(
    Widget  widget
    )
```

widget is the Enhanced Toggle widget.

XrtGearToggleButtonSetState()

Sets the current state of an Enhanced Toggle widget.

```
int
XrtGearToggleButtonGetState(
    Widget      widget,
    int         state,
    Boolean     notify
    )
```

widget is the Enhanced Toggle widget; *state* is the new state of the widget; *notify* indicates whether to call the callbacks in the `XmNxrtGearStateChangedCallback` list.

4

Tabbed Dialogs

Introduction ■ *Sample Program*
Widget Synopses ■ *Changing the Active Tab*
Attaching Pages to Tab Buttons ■ *Controlling Tab Display*
Constraint Resources ■ *Tab Button Resources*
Specifying Corner Size ■ *Controlling Resizing*
Resource Reference ■ *Procedures and Methods Reference*
Translations and Actions

4.1 Introduction

To program tabbed dialogs, you need to create the following:

- A Tab Manager widget to control the display and organization of tabs
- One or more Tab Button widgets to define tabs for your dialog
- Page widgets to be stored in the tabbed dialog

Together, the Tab Manager and Tab Buttons simulate a sequence of file folders. The page widgets are defined as children of the Tab Manager; a constraint resource is then used to associated a page widget with a particular Tab Button. Page widgets can be of any class (including any Manager widget class).

Figure 12 illustrates the anatomy of a typical tabbed dialog.

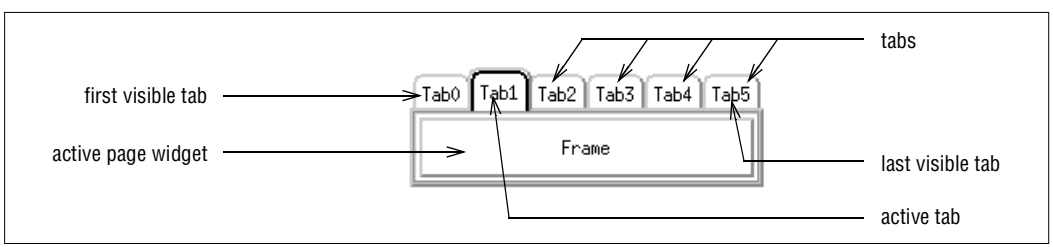


Figure 12 Anatomy of a tabbed dialog

4.2 Sample Program

The following code creates a Tab Manager widget, then defines several tabs and pages as its children:

```
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/XrtGLabel.h>
#include <Xm/Frame.h>
#include <Xm/XrtTabB.h>
#include <Xm/XrtTabM.h>

typedef struct my_widgets {
    Position offset;
    String name;
    Widget w;
} MyWidgets;

MyWidgets tab_table[] = {
        10, "Memos", NULL,
        60, "Faxes", NULL,
        260, "Postponed", NULL,
};

Widget page_table[3];

int
main(int argc, char **argv)
{
    XtAppContext app_context;
    Widget toplevel, manager, page;
    int i;
    XmString xms;

    toplevel = XtVaAppInitialize(
        &app_context, "Button",
        NULL, 0, (Cardinal *)&argc, argv, NULL);
    manager = XtVaCreateWidget( "tab_manager",
        xmXrtTabManagerWidgetClass, toplevel,
        NULL);
    xms = XmStringCreateSimple("Page widget");
    for (i = 0; i < XtNumber(tab_table); i++) {
        tab_table[i].w = XtVaCreateManagedWidget(tab_table[i].name,
            xmXrtTabButtonWidgetClass, manager,
            XmNxrtGearChildType, XRTGEAR_CHILD_TYPE_TAB,
            NULL);
        page_table[i] = XtVaCreateManagedWidget(tab_table[i].name,
            xmFrameWidgetClass, manager,
            XmNxrtGearChildType, XRTGEAR_CHILD_TYPE_PAGE,
            NULL);
        XtVaSetValues( tab_table[i].w,
            XmNxrtGearPageWidget, page_table[i],
            NULL);
        XtVaCreateManagedWidget("pb1",
            xmXrtGLabelWidgetClass, page_table[i],
            XmNlabelString, xms,
            NULL);
    }
    XmStringFree(xms);
```

```
    XtManageChild(manager);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app_context);
}
```

When this program is compiled and run, the following appears:



Figure 13 A sample Tab Manager with child Tab Buttons

This application consists of a Tab Manager, with three Tab Buttons defined as its children. Each Tab Button has a page associated with it. Each page, in turn, has a label widget defined as its child; in more complicated real-life applications, additional container widgets are defined as children of the pages.

To create a multi-level file folder, define another Tab Manager as a page widget.

4.3 Widget Synopses

Tab Button

Include File:	*<Xm/XrtTabB.h>*
Class Name:	XmXrtTabButton
Class Hierarchy:	Core → XmPrimitive → XmLabel → XmPushButton → XmXrtTabButton
Class Pointer:	xmXrtTabButtonWidgetClass

Tab Manager

Include File:	*<Xm/XrtTabM.h>*
Class Name:	XmXrtTabManager
Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtTabManager
Class Pointer:	xmXrtTabManagerWidgetClass

The Tab Manager widget is subclassed from the XmManager widget, and Tab Button is subclassed from XmPushButton.

For more information on the Motif widget hierarchy and where Tab Manager and Tab Button fit into this hierarchy, see section 1.2 on page 8.

4.4 Changing the Active Tab

The Tab Manager widget provides three resources that control the display of the tab manager and its children:

- `XmNxrtGearActivePageWidget` contains the currently active page widget.
- `XmNxrtGearActivePageNumber` contains the active page number.
- `XmNxrtGearActiveTabWidget` contains the currently active tab widget.

Page Changed Callbacks

The `XmNxrtGearPageCallback` resource defines callbacks to be invoked whenever the value of the `XmNxrtGearPageNumber` resource is set either interactively or programmatically[1]. This enables you to control page display.

Each `XmNxrtGearPageCallback` resource is passed the following structure:

```
typedef struct {
    int             reason;                         // Read-only
    XEvent          *event;                         // Read-only
    XrtGearAction   action;                         // Read-only
    int             prev_page_number;               // Read-only
    Widget          prev_page_widget;               // Read-only
    Widget          prev_tab;                       // Read-only
    Widget          tab;
    int             page_number;
    Widget          page_widget;
    Boolean         doit;
} XrtGearPageCallbackStruct;
```

Each callback is called both before and after the page change takes place. *reason* indicates which event is occurring, and is either `XRTGEAR_REASON_PAGE_BEGIN` (before page change) or `XRTGEAR_REASON_PAGE_END` (after page change).

The *prev_page_widget* and *prev_page_number* members enable you to access the previously displayed page widget and its number. The Tab Button associated with the previous page is specified by *prev_tab*. When the tab manager first appears, the page changed callbacks are invoked, and *prev_page_number* is -1.

The new page widget and its number are specified by *page_number* and *page_widget*. These members can be modified, which allows you to control page changes in a Tab Manager. *tab* is the Tab Button associated with the new page widget.

The *doit* field provides an easy way to suppress page changes. If *doit* is set to `False` when *reason* is `XRTGEAR_REASON_PAGE_BEGIN`, the page change does not occur.

1. `XmNxrtGearPageNumber` is changed whenever `XmNxrtGearActivePageWidget` and `XmNxrtGearActiveTabWidget` are set.

The following is an example of a page changed callback that prevents the end-user returning to the first page widget after leaving it:

```
void
pageCB(Widget w, XtPointer client_data,
XrtGearPageCallbackStruct *call_data)
{
    if (call_data->reason == XRTGEAR_REASON_PAGE_BEGIN &&
            call_data->page_number == 0 &&
            call_data->prev_page_number != -1) {
        call_data->doit = False;
    }
}
```

4.5 Attaching Pages to Tab Buttons

By default, if a Tab Button is created and a page is not explicitly assigned to it, the Tab Manager attaches the tab to any available page. This is convenient when your application is creating all its Tab Buttons and pages at the start of the program.

If your application is creating and destroying pages and Tab Buttons dynamically, you may not want to allow the Tab Manager to automatically attach Tab Buttons to pages. To insist that all attachments of Tab Buttons to pages must be explicitly specified by the application, set the `XmNxrtGearPageAttachPolicy` resource to `XRTGEAR_ATTACH_NONE`.

The default value for `XmNxrtGearPageAttachPolicy` is `XRTGEAR_ATTACH_ALWAYS`, which specifies automatic attachment when needed.

Note that if `XmNxrtGearPageAttachPolicy` is set after the Tab Manager has been created, it does not have any effect until the next Tab Button or page is created or destroyed.

The following callback (added to the list specified by `XmNxrtGearPageCallback`) creates pages dynamically:

```
void pageChange(Widget w, XtPointer client_data,
    XtPointer call_data)
{
    XrtGearPageCallbackStruct *cbstruct =
              (XrtGearPageCallbackStruct *) call_data;
    int         page_number;
    char        buff[80];
    Widget      page;

    if (cbstruct->reason == XRTGEAR_REASON_PAGE_END) {
       return;
    }
    if (cbstruct->page_widget == NULL) {
        XtVaGetValues(cbstruct->tab,
            XmNuserData, &page_number,
            NULL);
        sprintf( buff, "Page%02d", page_number);
        page = XtVaCreateManagedWidget(buff,
            xmLabelWidgetClass, w,
```

```
                XmNxrtGearChildType, XRTGEAR_CHILD_TYPE_PAGE,
                XmNxrtGearPageNumber, page_number,
                NULL);
            XtVaSetValues(cbstruct->tab,
                XmNxrtGearPageWidget, page,
                NULL);
            cbstruct->page_widget = page;
        }
    }
```

Setting the *page_widget* element of the callback structure forces the Tab Manager to display the new page.

4.6 Controlling Tab Display

Tab Manager enables you to control the following tab display features:

- The side of the Tab Manager on which tabs are displayed
- Which tabs are visible
- The spacing between tabs
- Tab resizing
- Tab rotation
- Tab Manager highlight thickness
- Tab Manager margin spacing
- Tab stretching
- Paging arrow button display

Display Side Specification

`XmNxrtGearTabSide` specifies the side on which tabs are to be displayed. It can be set to one of the following:

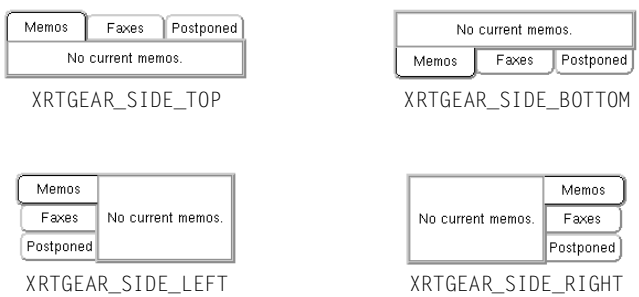


Figure 14 Tab button display sides

The default value is `XRTGEAR_SIDE_TOP`.

Tab Visibility

`XmNxrtGearFirstVisibleTab` controls the first tab currently visible (either leftmost or topmost). When this resource is set, the tabs are scrolled if necessary.

`XmNxrtGearLastVisibleTab` controls the last tab currently visible (either bottom-most or rightmost). When this resource is set, the tabs are scrolled if necessary.

To ensure that a particular tab is visible, call the *XrtGearTabManagerMakeTabVisible()* method. For example, the following call ensures that the second tab is visible (0 is the index of the first tab, 1 the second, and so on):

```
Widget      tabmgr;
Boolean     ok;

ok = XrtGearTabManagerMakeTabVisible(tabmgr, 1);
```

If the specified tab is to the left of the visible area, it becomes the leftmost visible tab after scrolling. If the specified tab is to the right of the visible area, it becomes the rightmost visible tab after scrolling.

XrtGearTabManagerMakeTabVisible() returns a Boolean value indicating whether the visible tab operation was successful. (It returns `False` if, for example, the tab index is invalid.)

Tab Spacing

The spacing between tabs (measured in pixels) is specified by the `XmNxrtGearTabSpacing` resource.

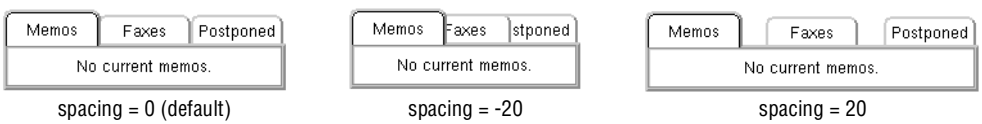


Figure 15 Tab spacing

When this resource is set to a negative value, the tabs will overlap.

Tab Resizing

The `XmNxrtGearTabResize` resource specifies whether all tabs should be resized to the width or height of the largest tab in the same row. If `XmNxrtGearTabResize` is `True` (the default value) resizing is performed; if `False`, tabs are not resized. If `XmNxrtGearTabSide` is set to `XRTGEAR_SIDE_TOP` or `XRTGEAR_SIDE_BOTTOM`, the width is used; otherwise, the height is used.



Figure 16 Tab resizing

To ensure that the tabs take up all the available visible space and are of the same size, set `XmNxrtGearTabResize` and `XmNxrtGearTabStretch` to `True`.

Tab Rotation

The `XmNxrtGearTabRotation` resource specifies whether tabs should be rotated. It can be set to any of the following:

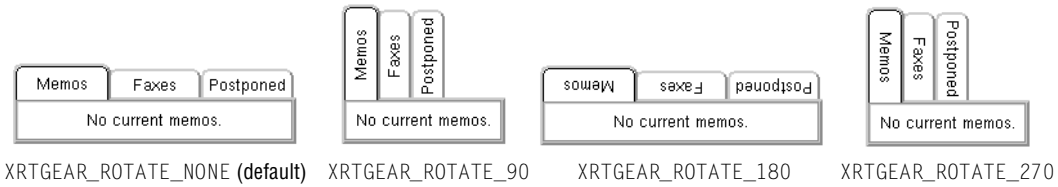


Figure 17 Tab rotation

As an alternative, you can use the Tab Button resources `XmNxrtGearPixmapRotation`, `XmNxrtGearStringRotation` and `XmNxrtGearStringLayout` to control the display of an individual Tab Button. These resources are described in section 4.8 on page 52.

Tab Manager Highlight Thickness

The `XmNxrtGearHighlightThickness` resource specifies the thickness of the Tab Manager's highlighting rectangle. By default, this thickness is 2 pixels.

Tab Manager Margins

The `XmNxrtGearMarginWidth` resource defines a left and right margin for the Tab Manager, which is the distance between the Tab Manager and its left and right neighbors. Similarly, the `XmNxrtGearMarginHeight` resource defines a top and bottom margin.

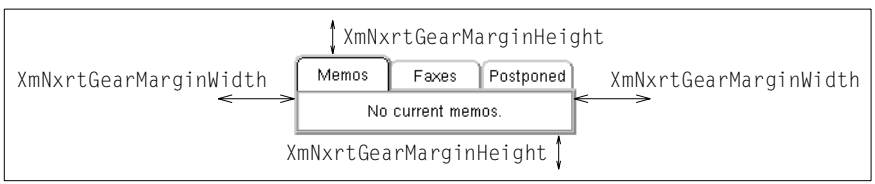


Figure 18 Tab Manager margin definitions

The default value for both resources is 2 pixels.

Tab Stretching

The `XmNxrtGearTabStretch` resource specifies whether all tabs should be stretched if necessary to fill the entire side.



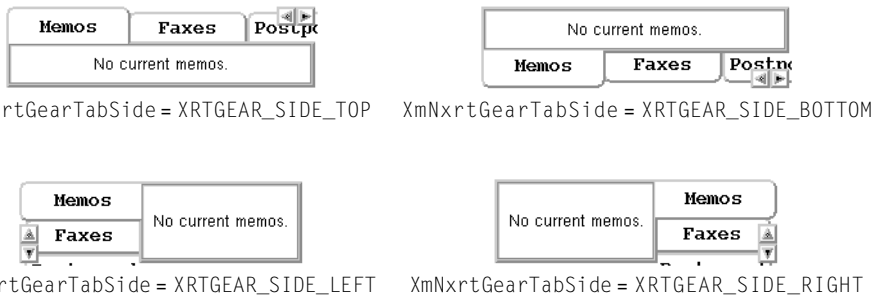
XmNxrtGearTabStretch = False XmNxrtGearTabStretch = True

Figure 19 Tab stretching

To ensure that the tabs take up all the available visible space and are of the same size, set XmNxrtGearTabResize and XmNxrtGearTabStretch to True.

Paging Arrow Button Display

By default, paging arrow buttons appear whenever the tab buttons cannot all be displayed. The location of the paging arrow buttons depends on the value of XmNxrtGearTabSide, as shown in Figure 20.[1]



XmNxrtGearTabSide = XRTGEAR_SIDE_TOP XmNxrtGearTabSide = XRTGEAR_SIDE_BOTTOM

XmNxrtGearTabSide = XRTGEAR_SIDE_LEFT XmNxrtGearTabSide = XRTGEAR_SIDE_RIGHT

Figure 20 Paging arrow buttons

The end-user scrolls the tab button display backward or forward by pressing the appropriate paging arrow button.

The XmNxrtGearArrowDisplayPolicy resource controls whether the paging arrow buttons appear. Valid values for this resource are:

XRTGEAR_DISPLAY_AS_NEEDED	display paging arrow buttons when needed (the default)
XRTGEAR_DISPLAY_ALWAYS	always display paging arrow buttons
XRTGEAR_DISPLAY_NEVER	never display paging arrow buttons

The height and width of the paging arrow buttons are controlled by the XmNxrtGearArrowHeight and XmNxrtGearArrowWidth resources, with the height and width specified in pixels. Default height and width behavior is as follows:

1. The arrow buttons appear on the opposite side of the tab manager if XmNstringDirection is set to XmSTRING_DIRECTION_R_TO_L.

- If neither resource is set, the height and width are set to half the tab button height.
- If XmNxrtGearArrowHeight is set and XmNxrtGearArrowWidth is not, XmNxrtGearArrowHeight is also used to specify the width.
- If XmNxrtGearArrowWidth is set and XmNxrtGearArrowHeight is not, XmNxrtGearArrowWidth is also used to specify the height.

A paging arrow button cannot have a height greater than the height of a tab button.

4.7 Constraint Resources

The following constraint resources are defined for children of the Tab Manager widget:

- XmNxrtGearChildType indicates whether a child is a page or a tab. If a child is a Tab Button widget, it is used as a tab, and its XmNxrtGearChildType value is XRTGEAR_CHILD_TYPE_TAB. All other widgets are assumed to be pages, and have an XmNxrtGearChildType value of XRTGEAR_CHILD_TYPE_PAGE.[1] (Currently, only Tab Button widgets are supported as tabs.)
- XmNxrtGearPageNumber specifies the page number of a page child. If this number is not set by the end-user, its default value is calculated using the order in which the pages were created, with the first page being given the number 0.

  For tab children, XmNxrtGearPageNumber specifies the page number of the page connected to the tab. If XmNxrtGearPageNumber is -1, the page widget specified by XmNxrtGearPageWidget (described below) is used.
- XmNxrtGearPageWidget specifies the page associated with a particular Tab Button widget. Several tabs may share a common page widget. This constraint resource can only be used with the Tab Button widget.

4.8 Tab Button Resources

The following sections describe how to use the Tab Button widget.

4.8.1 Specifying the Button Type

The XmNxrtGearLabelType resource controls the type of label used on the Tab Button. This resource can be set to any of the following:

XRTGEAR_TYPE_STRING	Button label is of type XmString; XmNlabelString is displayed.

1. Private, internal child widgets have an XmNxrtGearChildType value of XRTGEAR_CHILD_TYPE_INTERNAL. This value cannot be set by the application.

XRTGEAR_TYPE_PIXMAP	Button label is a pixmap; XmNxrtGearLabelPixmap is displayed.
XRTGEAR_TYPE_STRING_PIXMAP	Button label contains both an XmString and a pixmap; both XmNlabelString and XmNxrtGearLabelPixmap are displayed.

The default value for XmNxrtGearLabelType is XRTGEAR_TYPE_STRING.

4.8.2 Tab Anchoring

When a Tab Button is specified as a child of a Tab Manager, its placement is controlled by the Tab Manager's XmNxrtGearTabSide resource. If a Tab Button is defined as a child of any other widget, its location on its parent is specified by the XmNxrtGearAnchorSide resource. This can be set to any of the following values:

XRTGEAR_ANCHOR_SIDE_TOP	the top of the Tab Button is attached to the parent widget (default)
XRTGEAR_ANCHOR_SIDE_BOTTOM	the bottom of the Tab Button is attached to the parent widget
XRTGEAR_ANCHOR_SIDE_LEFT	the left edge of the Tab Button is attached to the parent widget
XRTGEAR_ANCHOR_SIDE_RIGHT	the right edge of the Tab Button is attached to the parent widget

4.8.3 Setting a Pixmap

To specify a pixmap for a Tab Button widget, set the XmNxrtGearLabelPixmap resource. This resource is of type Pixmap.

The pixmap specified by XmNxrtGearLabelPixmap is displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_PIXMAP or XRTGEAR_TYPE_STRING_PIXMAP.

The XPM and XBM pixmap formats are supported.

Using a Clip Mask

The XmNxrtGearMask resource allows you to define a clip mask for the pixmap displayed in a Tab Button widget. This clip mask controls what portion of the pixmap is to appear in the label. Only non-zero values in the mask are actually drawn. The following code creates a Tab Button widget that uses a clip mask:

```
Pixmap  image, mask;
Widget  button;

button = XtVaCreateManagedWidget("EnhLabel",
    xmXrtTabButtonWidgetClass,  tabmanager,
    XmNxrtGearChildType,        XRTGEAR_CHILD_TYPE_TAB,
    XmNxrtGearLabelType,        XRTGEAR_TYPE_PIXMAP,
    XmNxrtGearLabelPixmap,      image,
    XmNxrtGearMask,             mask,
    NULL);
```

XmNxrtGearMask expects values of type Pixmap. The default value for this resource is XmUNSPECIFIED_PIXMAP. The clip mask must be of depth 1. The same file can be used to specify a pixmap and its clip mask.

4.8.4 Controlling Button Shape

The shape of the Tab Button is controlled by two resources:

- XmNxrtGearStyle, which specifies the style of the button.
- XmNxrtGearCornerSize, which specifies the amount of rounding to be applied to the button.

XmNxrtGearStyle can be set to any of the following:

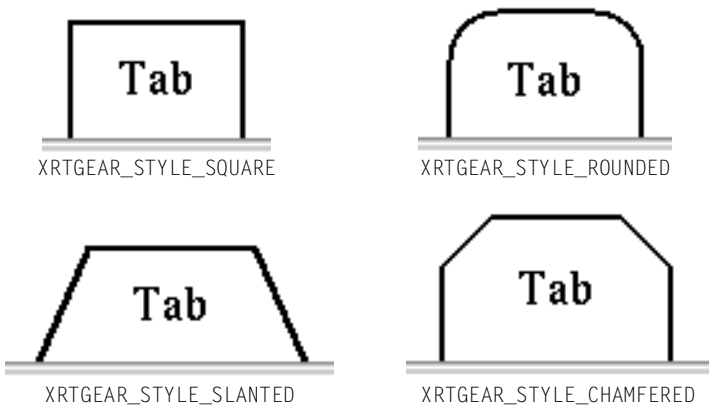


Figure 21 Tab button shapes

The default is XRTGEAR_STYLE_SQUARE. When XmNxrtGearStyle is set to XRTGEAR_STYLE_CHAMFERED, the corners are beveled.

Figure 22 shows the effects of various settings of XmNxrtGearCornerSize for different button types.

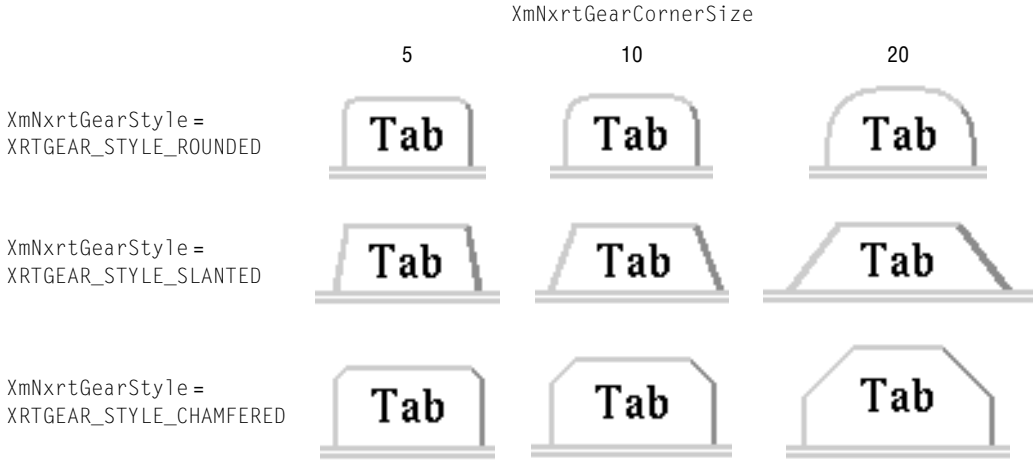


Figure 22 Corner sizes

If `XmNxrtGearStyle` is `XRTGEAR_STYLE_SQUARE`, `XmNxrtGearCornerSize` has no effect.

4.8.5 Text and Pixmap Positioning

When `XmNxrtGearLabelType` is set to `XRTGEAR_TYPE_STRING_PIXMAP`, both a text string and a pixmap are displayed in the Tab Button. Two resources are defined to control the relationship between the text string and the pixmap:

- `XmNxrtGearLayoutSpacing`, which specifies the distance in pixels between the text string and the pixmap (default is 2).
- `XmNxrtGearStringLayout`, which specifies the relative position of the text string with respect to the pixmap.

`XmNxrtGearStringLayout` can be set to any of the values shown in Figure 23.

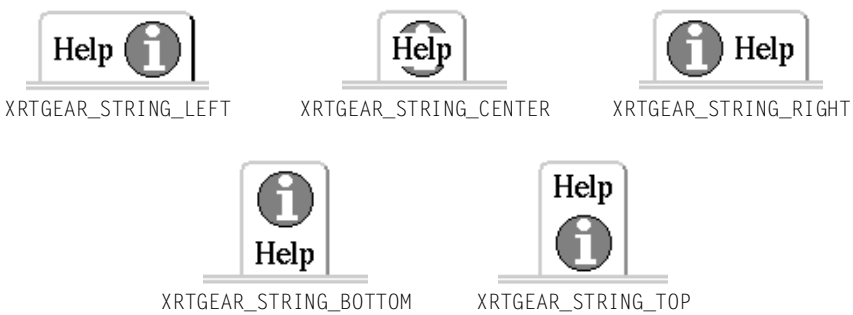


Figure 23 Tab Button text and pixmap layouts

The default is XRTGEAR_STRING_RIGHT.

XmNxrtGearLayoutSpacing and XmNxrtGearStringLayout are ignored unless XmNxrtGearLabelType is set to XRTGEAR_TYPE_STRING_PIXMAP.

If the Tab Button is a child of a Tab Manager whose XmNxrtGearTabRotation resource is set to something other than XRTGEAR_ROTATE_UNSPECIFIED, XmNxrtGearStringLayout is ignored.

Margin Control

The XmNmarginLeft, XmNmarginRight, XmNmarginTop, XmNmarginBottom, XmNmarginHeight and XmNmarginWidth resources (inherited from XmLabel) are used to calculate the space between the edge of the Tab Button and the string/pixmap:

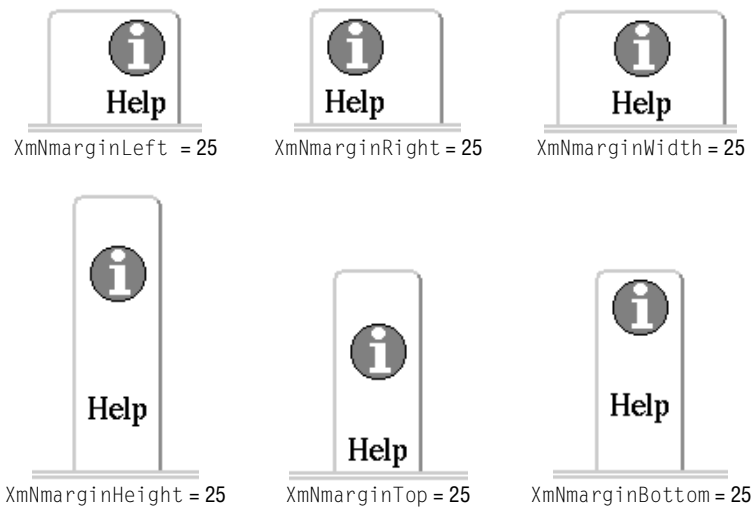


Figure 24 Tab Button margin control

4.8.6 Text and Pixmap Rotation

The text and pixmap in a Tab Button can be rotated separately. To rotate the Tab Button's text, set the XmNxrtGearStringRotation resource. This resource can be set to the values shown in Figure 25.

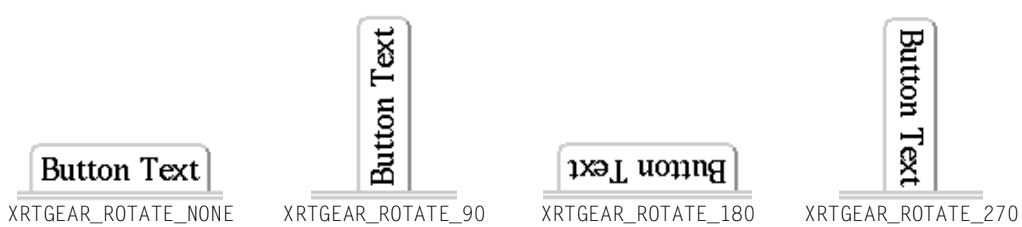


Figure 25 Tab Button text rotation

The default is XRTGEAR_ROTATE_NONE. All rotations are counterclockwise.

If the Tab Button is a child of a Tab Manager whose XmNxrtGearTabRotation resource is set to something other than XRTGEAR_ROTATE_UNSPECIFIED, XmNxrtGearStringRotation is ignored.

To rotate a Tab Button pixmap, set the XmNxrtGearPixmapRotation resource. This resource can be set to the same values as XmNxrtGearStringRotation:

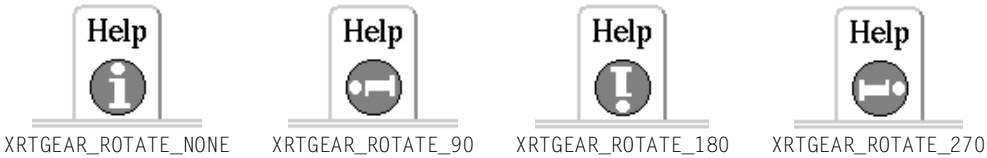


Figure 26 Tab Button pixmap rotation

The default for XmNxrtGearPixmapRotation is also XRTGEAR_ROTATE_NONE.

If the Tab Button is a child of a Tab Manager whose XmNxrtGearTabRotation resource is set, XmNxrtGearPixmapRotation is ignored.

4.8.7 Specifying Tab Button and Page Shadow Colors

Each Tab Button widget obtains its color specifications from the page widget with which it is associated (specified by the XmNxrtGearPageWidget resource). To specify your own Tab Button colors, change the appropriate color resources (such as XmNforeground) *after* associating a page widget with the Tab Button.

The shadow colors for each page widget are, in turn, obtained from the Tab Button color specifications.

4.9 Specifying Corner Size

If XmNxrtGearStyle is set to anything other than XRTGEAR_STYLE_SQUARE, the amount of rounding, slanting or beveling performed on each of the Tab Button corners is specified by the Tab Manager's XmNxrtGearCornerSize resource. If XmNxrtGearStyle is set to XRTGEAR_STYLE_CHAMFERED (which specifies beveling) or XRTGEAR_STYLE_ROUNDED, XmNxrtGearCornerSize specifies the width and height of the corner:

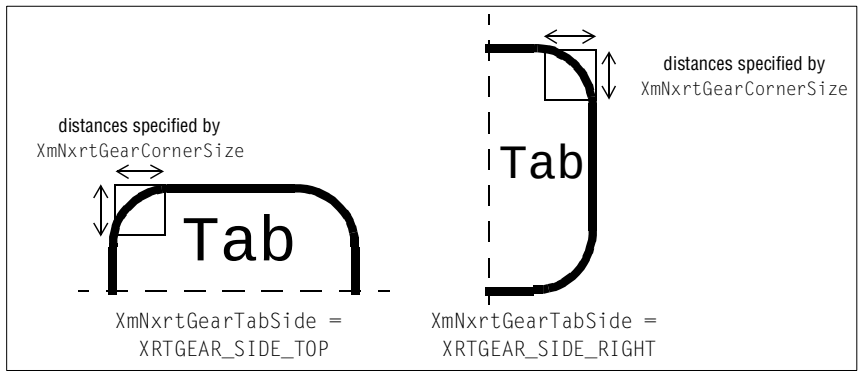


Figure 27 Corner size of a rounded or beveled tab button

If tab corners are slanted (`XmNxrtGearStyle` is `XRTGEAR_STYLE_SLANTED`), `XmNxrtGearCornerSize` specifies the amount of slanting to perform:

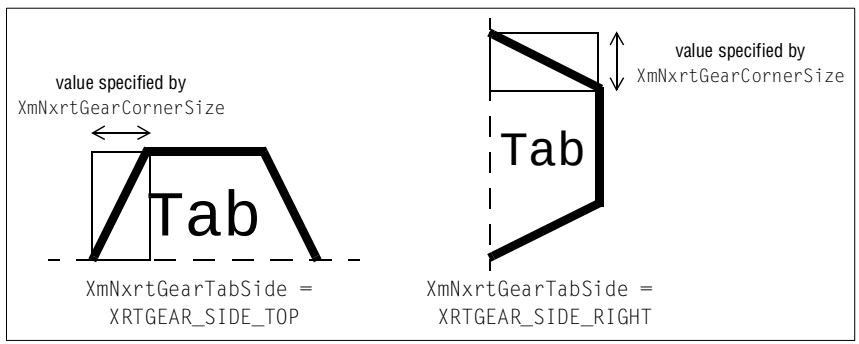


Figure 28 Corner size of a slanted tab button

If `XmNxrtGearCornerSize` is set to a value larger than the tab width/height, 0 is used.

`XmNxrtGearCornerSize` is also defined as a Tab Button resource, for use when the Tab Button is not a child of the tab manager. If the Tab Button is a child of a Tab Manager, the value of the Tab Button `XmNxrtGearCornerSize` resource is only used if it is set after the Tab Manager is managed.

4.10 Controlling Resizing

The `XmNxrtGearResizeCallback` Tab Manager resource enables you to supply callbacks to be invoked whenever the Tab Manager is resized.

Each Resize callback is passed the following structure:

```
typedef struct {
    int         reason;     /* read-only */
    XEvent      *event;     /* read-only */
    Dimension   width;
    Dimension   height;
} XrtGearResizeCallbackStruct;
```

The *width* and *height* members of this structure specify the new height and width of the Tab Manager.

4.11 Resource Reference

This section lists all of the resources for the Tab Button and Tab Manager widgets, plus some of the more commonly used inherited resources. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

4.11.1 Tab Button Resources

XmNxrtGearAnchorSide **XrtGearAnchorSide** **XRTGEAR_ANCHOR_SIDE_TOP** **CSG**

Specifies the side of the widget that is to be joined to the parent: `XRTGEAR_ANCHOR_SIDE_LEFT`, `XRTGEAR_ANCHOR_SIDE_RIGHT`, `XRTGEAR_ANCHOR_SIDE_TOP` or `XRTGEAR_ANCHOR_SIDE_BOTTOM`. This resource should only be set if the Tab Button's parent is not a Tab Manager.

XmNbackground **Pixel** ***dynamic*** **CSG**

Specifies the background color of the button.

XmNxrtGearCornerSize **Dimension** ***dynamic*** **CSG**

Specifies the amount of rounding, slanting or beveling to apply to the corners of the tab button. If `XmNxrtGearStyle` is `XRTGEAR_STYLE_CHAMFERED` (which specifies beveling) or `XRTGEAR_STYLE_ROUNDED`, `XmNxrtGearCornerSize` specifies the width and height of the corner in pixels. If `XmNxrtGearStyle` is `XRTGEAR_STYLE_SLANTED`, `XmNxrtGearCornerSize` specifies the amount of slanting in pixels.

`XmNxrtGearCornerSize` is ignored if `XmNxrtGearStyle` is `XRTGEAR_STYLE_SQUARE`.

If a Tab Button is the child of a Tab Manager, the Tab Manager's `XmNxrtGearCornerSize` resource is used (unless the Tab Button's resource is set after the Tab Manager is managed).

XmNfontList **XmFontList** ***dynamic*** **CSG**

Specifies the font list used for the widget text.

XmNforeground **Pixel** ***dynamic*** **CSG**

Specifies the foreground color of the button.

XmNheight **Dimension** ***dynamic*** **CSG**

Specifies the height of the window in pixels, excluding the border.

XmNxrtGearLabelPixmap **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies a label pixmap, displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_PIXMAP or XRTGEAR_TYPE_STRING_PIXMAP. XmNxrtGearLabelPixmap and XmNxrtGearMask can use the same XPM file.

XmNlabelString **XmString** ***dynamic*** **CSG**

Specifies a label string, displayed when XmNxrtGearLabelType is XRTGEAR_TYPE_STRING or XRTGEAR_TYPE_STRING_PIXMAP.

XmNxrtGearLabelType **XrtGearType** **XRTGEAR_TYPE_STRING** **CSG**

Specifies whether a text string, pixmap, or both are displayed. Valid values are XRTGEAR_TYPE_STRING (XmString), XRTGEAR_TYPE_PIXMAP (Pixmap) or XRTGEAR_TYPE_STRING_PIXMAP. If XmNlabelType is set to XmSTRING or XmPIXMAP, it will set the corresponding XmNxrtGearLabelType resource. If both XmNlabelType and XmNxrtGearLabelType are set concurrently, XmNxrtGearLabelType will take precedence.

XmNxrtGearLayoutSpacing **Dimension** **2** **CSG**

Specifies the number of pixels between the text and pixmap. This resource is only used if XmNxrtGearLabelType is XRTGEAR_TYPE_STRING_PIXMAP.

XmNmarginHeight **Dimension** **2** **CSG**

Specifies the space between the top and bottom edges of the button and the nearest shadow edge.

XmNmarginWidth **Dimension** **2** **CSG**

Specifies the space between the left and right edges of the button and the nearest shadow edge.

XmNxrtGearMask **Pixmap** **XmUNSPECIFIED_PIXMAP** **CSG**

Specifies the clip mask to be used. Only non-zero values in the mask are drawn. The clip mask must be of depth 1. XmNxrtGearLabelPixmap and XmNxrtGearMask can use the same XPM file.

XmNxrtGearPixmapRotation **XrtGearRotation** **XRTGEAR_ROTATE_NONE** **CSG**

Specifies the rotation of the pixmap. Valid values are: XRTGEAR_ROTATE_NONE, XRTGEAR_ROTATE_90, XRTGEAR_ROTATE_180, or XRTGEAR_ROTATE_270. Rotation is measured counterclockwise.

If the Tab Button is a child of a Tab Manager, the Tab Manager's XmNxrtGearTabRotation resource controls the value of this resource if set.

XmNxrtGearStringLayout **XrtGearStringLayout** **XRTGEAR_STRING_RIGHT** **CSG**

Specifies the relative position of the text with respect to the pixmap. Valid values are: XRTGEAR_STRING_CENTER, XRTGEAR_STRING_TOP, XRTGEAR_STRING_LEFT, XRTGEAR_STRING_BOTTOM or XRTGEAR_STRING_RIGHT.

If the Tab Button is a child of a Tab Manager, the Tab Manager's XmNxrtGearTabRotation resource controls the value of this resource if set.

This resource is ignored if XmNxrtGearLabelType is not XRTGEAR_TYPE_STRING_PIXMAP.

XmNxrtGearStringRotation **XrtGearRotation** **XRTGEAR_ROTATE_NONE** **CSG**

Specifies the rotation of the text. Valid values are: XRTGEAR_ROTATE_NONE, XRTGEAR_ROTATE_90, XRTGEAR_ROTATE_180, or XRTGEAR_ROTATE_270. Rotation is measured counterclockwise.

If the Tab Button is a child of a Tab Manager, the Tab Manager's XmNxrtGearTabRotation resource controls the value of this resource if set.

XmNxrtGearStyle **XrtGearStyle** **XRTGEAR_STYLE_ROUNDED** **CSG**

Specifies the tab corner style to use. Valid values are: XRTGEAR_STYLE_ROUNDED, XRTGEAR_STYLE_SQUARE, XRTGEAR_STYLE_SLANTED or XRTGEAR_STYLE_CHAMFERED.

XmNuserData **XtPointer** **NULL** **CSG**

Specifies a pointer to application-defined data.

XmNwidth **Dimension** ***dynamic*** **CSG**

Specifies the width of the window in pixels, excluding the border.

XmNx **Position** **0** **CSG**

The X-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

XmNy **Position** **0** **CSG**

The Y-coordinate of the top-left outer corner of the widget, relative to the top-left inner corner of its parent widget.

4.11.2 Tab Manager Resources

XmNxrtGearActivePageNumber **int** ***dynamic*** **CSG**

Specifies the active page number. If a non-existent page number is set, a blank page is displayed. This resource may also be set to XRTGEAR_PAGE_FIRST or XRTGEAR_PAGE_LAST. If both the XmNxrtGearActivePageWidget and XmNxrtGearActivePageNumber are set concurrently, the XmNxrtGearActivePageWidget value has precedence.

XmNxrtGearActivePageWidget **Widget** ***dynamic*** **CSG**

Specifies the current page widget. If a NULL page is set, a blank page is displayed. If both the XmNxrtGearActivePageWidget and XmNxrtGearActivePageNumber are set concurrently, the XmNxrtGearActivePageWidget value has precedence.

XmNxrtGearActiveTabWidget **Widget** ***dynamic*** **CSG**

Specifies the currently active tab widget. If a NULL tab is set, a blank page is displayed. If both the XmNxrtGearActivePageWidget and XmNxrtGearActivePageNumber are set concurrently with this resource, the XmNxrtGearActivePageWidget value has precedence.

XmNxrtGearArrowDisplayPolicy **XrtGearDisplayPolicy** **XRTGEAR_DISPLAY_AS_NEEDED** **CSG**

Specifies whether paging arrow buttons are to be displayed. Valid values are:

XRTGEAR_DISPLAY_AS_NEEDED	display paging arrow buttons when needed
XRTGEAR_DISPLAY_ALWAYS	always display paging arrow buttons
XRTGEAR_DISPLAY_NEVER	never display paging arrow buttons

XmNxrtGearArrowHeight **Dimension** **11** **CSG**

Specifies the height of the paging arrow buttons.

If this resource is not set but XmNxrtGearArrowWidth is set, the height is set to the value of XmNxrtGearArrowWidth. If neither resource is set, the height and width are set to half the height of a tab button.

A paging arrow button cannot have a height greater than the height of the tab buttons.

XmNxrtGearArrowWidth **Dimension** **11** **CSG**

Specifies the width of the paging arrow buttons.

If this resource is not set but XmNxrtGearArrowHeight is set, the width is set to the value of XmNxrtGearArrowHeight. If neither resource is set, the height and width are set to half the height of a tab button.

XmNxrtGearPageAttachPolicy **XrtGearPageAttachPolicy** **XRTGEAR_ATTACH_ALWAYS** **CSG**

Specifies whether a Tab Button is to be automatically attached to any available page if no attachment is explicitly specified. Valid values are:

XRTGEAR_ATTACH_ALWAYS	Always automatically attach when necessary
XRTGEAR_ATTACH_NONE	Never automatically attach a Tab Button to a page; all attachments must be explictly specified

Note that if XmNxrtGearPageAttachPolicy is set after the Tab Manager has been created, it does not have any effect until the next Tab Button or page is created or destroyed.

XmNxrtGearCornerSize **Dimension** **5** **CSG**

Specifies the amount of rounding, slanting or beveling to apply to the corners of the tab button. If XmNxrtGearStyle is XRTGEAR_STYLE_CHAMFERED (which specifies beveling) or XRTGEAR_STYLE_ROUNDED, XmNxrtGearCornerSize specifies the width and height of the corner in pixels. If XmNxrtGearStyle is XRTGEAR_STYLE_SLANTED, XmNxrtGearCornerSize specifies the amount of slanting in pixels.

XmNxrtGearCornerSize is ignored if XmNxrtGearStyle is XRTGEAR_STYLE_SQUARE.

XmNxrtGearFirstVisibleTab **int** ***dynamic*** **CSG**

Specifies the first tab currently visible (either left-most or top-most). Setting this resource causes the tabs to scroll if necessary so that the tab is fully visible.

XmNhighlightThickness **Dimension** **2** **CSG**

Specifies the thickness of the highlighting rectangle.

XmNxrtGearLastVisibleTab **int** ***dynamic*** **CSG**

Specifies the last tab currently visible (either bottom-most or right-most). Setting this resource causes the tabs to scroll if necessary so that the tab is fully visible.

XmNxrtGearMarginHeight **Dimension** **2** **CSG**

Specifies the Tab Manager's top and bottom margins (the space on the top and bottom sides of the Tab Manager).

XmNxrtGearMarginWidth **Dimension** **2** **CSG**

Specifies the Tab Manager's left and right margins (the space on the left and right sides of the Tab Manager).

XmNxrtGearPageCallback **XtCallbackList** **NULL** **CSG**

Specifies the list of callbacks called when the `XmNxrtGearPageNumber` is changed, either interactively or programatically. Each routine is passed a pointer to an `XrtGearPageCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an XrtGearPageCallbackStruct */
)
```

where `XrtGearPageCallbackStruct` is defined as:

```
typedef struct {
    int             reason;             // Read-only
    XEvent          *event;             // Read-only
    XrtGearAction   action;             // Read-only
    int             prev_page_number;   // Read-only
    Widget          prev_page_widget;   // Read-only
    Widget          prev_tab;           // Read-only
    Widget          tab;
    int             page_number;
    Widget          page_widget;
    Boolean         doit;
} XrtGearPageCallbackStruct;
```

reason is `XRTGEAR_REASON_PAGE_BEGIN` or `XRTGEAR_REASON_PAGE_END`. *event* is the XEvent that triggered the callback, or `NULL` if the callback is called programmatically. *action* is `XRTGEAR_ACTION_SET_VALUE`. *prev_page_number* is the previous page number (`XRTGEAR_NOVALUE` when a blank page is displayed). *prev_page_widget* is the previous page widget (`NULL` at initialization). *prev_tab* is the previous tab widget (`NULL` at initialization). *tab* is the new tab widget. *page_number* and *page_widget* are the page number and widget to be displayed, and

may be changed; if both are changed, *page_widget* takes precedence. *doit* should be set to `False` if *reason* is `_BEGIN` and the page is not to change.

XmNxrtGearResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies the list of callbacks called when the Tab Manager is resized. Each routine is passed a pointer to an `XrtGearResizeCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an XrtGearResizeCallbackStruct */
)
```

where `XrtGearResizeCallbackStruct` is defined as:

```
typedef struct {
    int             reason;         // Read-only
    XEvent          *event;         // Read-only
    Dimension       width;
    Dimension       height;
} XrtGearResizeCallbackStruct;
```

reason is `XRTGEAR_REASON_RESIZE`. *event* is the XEvent that triggered the callback, or `NULL` if the callback is called programmatically. *width* and *height* are the new width and height of the Tab Manager.

XmNxrtGearShadowThickness **Dimension** **2** **CSG**

Specifies the thickness of the shadow border.

XmNxrtGearTabResize **Boolean** **True** **CSG**

Specifies whether all tabs should be resized to the width or height of the largest tab in the same row, depending on the value of the `XmNxrtGearTabSide` resource. If `True`, this resource is processed before `XmNxrtGearTabStretch`.

XmNxrtGearTabRotation **XrtGearTabRotation** **XRTGEAR_ROTATE_UNSPECIFIED** **CSG**

Specifies the value for the Tab Button's XmNxrtGearPixmapRotation, XmNxrtGearStringRotation and XmNxrtGearStringLayout resources. XmNxrtGearTabRotation can be set to one of the following:

XRTGEAR_ROTATE_UNSPECIFIED	No rotation specified (the application cannot set the resource to this value)
XRTGEAR_ROTATE_NONE	Do not rotate
XRTGEAR_ROTATE_90	Rotate 90 degrees counterclockwise
XRTGEAR_ROTATE_180	Rotate 180 degrees counterclockwise
XRTGEAR_ROTATE_270	Rotate 270 degrees counterclockwise

The following table shows the relationship between the above XmNxrtGearTabRotation resource values and the Tab Button resources they affect:

if you set XmNxrtGearTabRotation **to this value**	XmNxrtGearPixmapRotation **and** XmNxrtGearStringRotation **are set to this value**	**and** XmNxrtGearStringLayout **is set to this value**
XRTGEAR_ROTATE_NONE	XRTGEAR_ROTATE_NONE	XRTGEAR_STRING_RIGHT
XRTGEAR_ROTATE_90	XRTGEAR_ROTATE_90	XRTGEAR_STRING_TOP
XRTGEAR_ROTATE_180	XRTGEAR_ROTATE_180	XRTGEAR_STRING_LEFT
XRTGEAR_ROTATE_270	XRTGEAR_ROTATE_270	XRTGEAR_STRING_BOTTOM

If XmNxrtGearTabRotation is set to its default value of XRTGEAR_ROTATE_UNSPECIFIED, the Tab Button resources are left unchanged.

XmNxrtGearTabSide **XrtGearTabSide** **XRTGEAR_SIDE_TOP** **CSG**

Specifies the side on which tabs are to be displayed: XRTGEAR_SIDE_TOP, XRTGEAR_SIDE_BOTTOM, XRTGEAR_SIDE_LEFT, or XRTGEAR_SIDE_RIGHT.

XmNxrtGearTabSpacing **int** **0** **CSG**

Specifies the spacing between tabs in pixels. When set to a negative value, the tabs overlap.

XmNxrtGearTabStretch **Boolean** **False** **CSG**

Specifies whether all tabs should be stretched if necessary to fill the entire side. This resource is ignored when scrolling tabs.

4.11.3 Tab Manager Constraint Resources

XmNxrtGearChildType **XrtGearChildType** **XRTGEAR_CHILD_TYPE_PAGE** **G**

Specifies the type of a child of a Tab Manager widget. Any child which is a Tab Button widget is used as a tab. All other widget classes are pages.

XRTGEAR_CHILD_TYPE_PAGE	When child is not a TabButton

XRTGEAR_CHILD_TYPE_TAB	When child is a Tab Button
XRTGEAR_CHILD_TYPE_INTERNAL	When child is a private, internal widget

XmNxrtGearPageNumber **int** ***dynamic*** **CSG**

For a page widget, this resource specifies the page number associated with this page widget. The default number is calculated from the order in which the pages were created, from 0. This resource is ignored if this widget's XmNxrtGearChildType value is not XRTGEAR_TYPE_PAGE.

For a tab widget, this resource specifies the number of the page connected to the tab. If this resource is set to -1, the page specified by XmNxrtGearPageWidget is used.

XmNxrtGearPageWidget **Widget** **NULL** **CSG**

Specifies the page associated with this tab widget. Several tabs may share a common page widget. This resource is ignored if this widget's XmNxrtGearChildType value is XRTGEAR_TYPE_PAGE.

XmNxrtGearWidgetPosition **int** ***dynamic*** **G**

Indicates the order of creation of the Tab Manager's children. The first child created has an XmNxrtGearWidgetPosition value of 0, the second child 1, and so on. This value cannot be set by the application.

4.11.4 Resources Inherited by Tab Button

Resource	Inherited From	Resource	Inherited From
XmNaccelerator	XmLabel	XmNinitialResourcesPersistent	Core
XmNaccelerators	Core	XmNlabelPixmap	XmLabel
XmNactivateCallback	XmPushButton	XmNlabelString	XmLabel
XmNalignment	XmLabel	XmNmappedWhenManaged	Core
XmNancestorSensitive	Core	XmNmarginBottom	XmLabel
XmNarmCallback	XmPushButton	XmNmarginHeight	XmLabel
XmNarmColor[a]	XmPushButton	XmNmarginLeft	XmLabel
XmNarmPixmap	XmPushButton	XmNmarginRight	XmLabel
XmNbackground	Core	XmNmarginTop	XmLabel
XmNbackgroundPixmap	Core	XmNmarginWidth	XmLabel
XmNborderColor[a]	Core	XmNmnemonic[b]	XmLabel
XmNborderPixmap	Core	XmNmnemonicCharSet[b]	XmLabel
XmNborderWidth	Core	XmNmultiClick	XmPushButton
XmNbottomShadowColor[a]	Primitive	XmNnavigationType	Primitive
XmNbottomShadowPixmap	Primitive	XmNrecomputeSize	XmLabel

Resource	Inherited From
XmNcolormap	Core
XmNdefaultButtonShadowThickness	XmPushButton
XmNdepth	Core
XmNdestroyCallback	Core
XmNdisarmCallback	XmPushButton
XmNfillOnArm	XmPushButton
XmNfontList	XmLabel
XmNforeground	Primitive
XmNheight	Core
XmNhelpCallback	Primitive
XmNhighlightColor[a]	Primitive
XmNhighlightOnEnter	Primitive
XmNhighlightPixmap	Primitive
XmNhighlightThickness	Primitive

Resource	Inherited From
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness[c]	Primitive
XmNshowAsDefault	XmPushButton
XmNstringDirection	XmLabel
XmNtopShadowColor[a]	Primitive
XmNtopShadowPixmap	Primitive
XmNtranslations	Core
XmNtraversalOn	Primitive
XmNuserData	Primitive
XmNwidth	Core
XmNx	Core
XmNy	Core

a. A Tab Button widget inherits its color resources from its associated page widget.
b. Only supported if `XmNxrtGearStringRotation` is `XRTGEAR_ROTATE_NONE`.
c. If a Tab Button is a child of a Tab Manager, the Tab Manager's `XmNshadowThickness` setting is used.

4.11.5 Resources Inherited by Tab Manager

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNinsertPosition	Composite
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNuserData	XmManager
XmNwidth	Core
XmNx	Core
XmNy	Core

4.12 Procedures and Methods Reference

This section lists the Tab Button and Tab Manager procedures and methods in alphabetical order.

XmCreateXrtTabButton()
Creates a Tab Button widget.

```
Widget
XmCreateXrtTabButton(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmCreateXrtTabManager()

Creates a Tab Manager widget.

```
Widget
XmCreateXrtTabManager(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmIsXrtTabButton()

Returns `True` if a widget is a Tab Button.

```
Boolean
XmIsXrtTabButton(
    Widget  widget
    )
```

XmIsXrtTabManager()

Returns `True` if a widget is a Tab Manager.

```
Boolean
XmIsXrtTabManager(
    Widget  widget
    )
```

XrtGearMrmInitialize()

Registers the XRT/gear widgets with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
void
XrtGearMrmInitialize(void)
```

XrtGearSetXmStringConverter()

Sets the XmRXmString resource converter to create XmString values.

```
void
XrtGearSetXmStringConverter(void)
```

The conversion uses the following syntax to specify embedded font changes (tags):

@font@	font's tag in `XmNfontList`
@P@	previous font
\@	a literal "@"

For example,

```
For a @bold@free evaluation@P@ of any product
```

with the fontlist:

```
fixed,-*-times-bold-r-normal-*-140-*=bold
```

displays:

```
For a free evaluation of any product
```

XrtGearTabManagerMakeTabVisible()

Scrolls the tabs of a Tab Manager widget (if necessary) to make the specified tab visible. If the tab was to the left of the visible area, it becomes the leftmost visible tab after scrolling; if the tab was to the right of the visible area, it becomes the rightmost visible tab after scrolling. Returns `True` if the scroll was successful.

```
Boolean
XrtGearTabManagerMakeTabVisible(
    Widget  widget,
    int     tab_num
    )
```

widget is the Tab Manager widget; *tab_num* is the index of the specified tab (0 specifies the first tab, 1 the second, and so on).

4.13 Translations and Actions

4.13.1 Default Tab Button Translations

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	XrtArm()
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>(2+)	MultiArm()
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	Activate() Disarm()
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>(2+)	MultiActivate()
<Key>osfSelect	ArmAndActivate()
<Key>osfActivate	ArmAndActivate()
<Key>osfHelp	Help()
Ctrl ~Alt ~Meta ~Shift <Key>Return	XrtArm() ArmAndActivate()
Ctrl ~Alt ~Meta ~Shift <Key>Space	XrtArm() ArmAndActivate()

4.13.2 Tab Button Actions

Activate()
Invokes the list of callbacks specified by `XmNactivateCallback`.

ArmAndActivate()
Invokes the list of callbacks specified by `XmNarmCallback`, `XmNactivateCallback`, and `XmNdisarmCallback`.

Disarm()
Invokes the list of callbacks specified by `XmNdisarmCallback`.

Help()
Restores the previous keyboard focus and invokes the callbacks specified by `XmNhelpCallback`.

MultiActivate()
Increments the *click_count* member of `XmPushButtonCallbackStruct`, and invokes the callbacks specified by `XmNactivateCallback` and `XmNdisarmCallback`.

This action routine only takes effect when the `XmNmultiClick` resource is set to `XmMULTICLICK_KEEP`.

MultiArm()
Invokes the list of callbacks specified by `XmNarmCallback`.

This action routine only takes effect when the `XmNmultiClick` resource is set to `XmMULTICLICK_KEEP`.

XrtArm()
Invokes the list of callbacks specified by `XmNarmCallback`.

5

The Toolbar Widget

Sample Program ■ *Widget Synopsis*
Specifying Orientation ■ *Margin and Spacing*
Constraint Resources ■ *Adding Help Balloons*
Resource Reference ■ *Procedures and Methods Reference*

5.1 Sample Program

The Toolbar widget is a simple Manager which contains buttons or widgets in a single row or column. It is typically used as a quick alternative to the menu bar.

The following code shows one way to create a Toolbar widget:

```
Pixmap cut_pixmap, cut_mask, copy_pixmap, copy_mask,
       paste_pixmap, paste_mask;
Widget column, toolbar, button1, button2, button3;

column = XtVaCreateWidget("column",
    xmRowColumnWidgetClass,     toplevel,
    XmNorientation,             XmVERTICAL,
    NULL);
toolbar = XtVaCreateWidget("toolbar",
    xmXrtToolbarWidgetClass,    column,
    NULL);
button1 = XtVaCreateManagedWidget("Cut",
    xmXrtPushButtonWidgetClass, toolbar,
    XmNlabelPixmap,             cut_pixmap,
    XmNxrtGearMask,             cut_mask,
    XmNxrtGearLabelType,        XRTGEAR_TYPE_PIXMAP,
    NULL);
button2 = XtVaCreateManagedWidget("Copy",
    xmXrtPushButtonWidgetClass, toolbar,
    XmNlabelPixmap,             copy_pixmap,
    XmNxrtGearMask,             copy_mask,
    XmNxrtGearLabelType,        XRTGEAR_TYPE_PIXMAP,
    NULL);
button3 = XtVaCreateManagedWidget("Paste",
    xmXrtPushButtonWidgetClass, toolbar,
    XmNlabelPixmap,             paste_pixmap,
    XmNxrtGearMask,             paste_mask,
```

```
        XmNxrtGearLabelType,        XRTGEAR_TYPE_PIXMAP,
        NULL);
    XtManageChild(toolbar);
    XtManageChild(column);
```

As each child is added, it is automatically positioned in the Toolbar from left to right.

Here is the Toolbar produced by the above code:



Figure 29 A sample Toolbar

5.2 Widget Synopsis

Include File:	*<Xm/XrtToolbar.h>*
Class Name:	XmXrtToolbar
Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtManager → XmXrtToolbar
Class Pointer:	xmXrtToolbarWidgetClass

The Toolbar widget is subclassed from the XmXrtManager widget, which is in turn subclassed from the XmManager widget.

For more information on the Motif widget hierarchy and where Enhanced Label and Enhanced Pushbutton fit into this hierarchy, see section 1.2 on page 8.

5.3 Specifying Orientation

By default, a toolbar is displayed as a row of widgets/icons. To display a toolbar as a column, set `XmNxrtGearOrientation` to `XmVERTICAL`.



Figure 30 Toolbar orientation

5.4 Margin and Spacing

The `XmNxrtGearMarginHeight` resource specifies the amount of space between the top and bottom sides of the Toolbar's edge and its children. The `XmNxrtGearMarginWidth` resource specifies the amount of space between the left and right sides of the Toolbar's edge and its children.



Figure 31 Toolbar margin resources

By default, both of these resources are set to 2.

5.5 Constraint Resources

The following Toolbar constraint resources specify the space between a child widget and its parent Toolbar:

- `XmNxrtGearBottomSpace` specifies the space between the widget and the item below it.
- `XmNxrtGearTopSpace` specifies the space between the widget and the item above it.
- `XmNxrtGearLeftSpace` specifies the space between the widget and its left neighbor.

- XmNxrtGearRightSpace specifies the space between the widget and its right neighbor.

Figure 32 shows a toolbar whose first child has had all of the above resources set to 20:

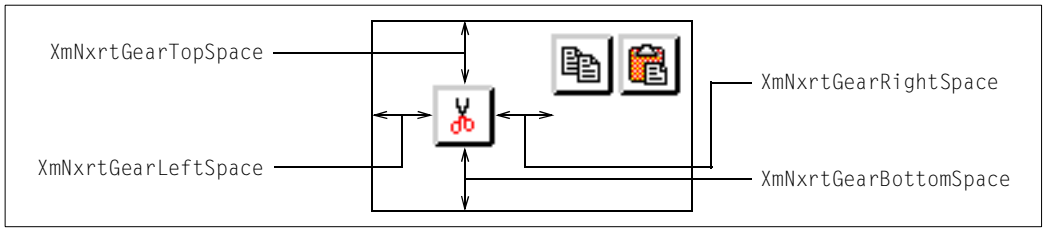


Figure 32 Toolbar constraint resources

Note that the margin resource orientation does not change when the Toolbar orientation changes: for example, XmNxrtGearLeftSpace still measures the distance between the child widget and its left neighbor.

Also note that the effects of Toolbar constraint resources are cumulative. For example, if one child has its XmNxrtGearRightSpace resource set to 20, and its right neighbor has its XmNxrtGearLeftSpace resource set to 10, the total distance between the two children is 30 pixels.

Specifying the Child Widget Position

XmNxrtGearPosition specifies the position of the child widget in the list of children. Changing the value of this resource changes the widget's visual position if the widget is managed. Setting XmNxrtGearPosition to 0 or XmLAST_POSITION moves the widget to the beginning or the end of the list.

5.6 Adding Help Balloons

To add help balloons to your toolbar, use Widget Tips, described in Chapter 11, "The Widget Tips Utility". (Widget Tips can be added to any widget, including Manager widgets.)

5.7 Resource Reference

This section lists all of the resources for the Toolbar widget. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

XmNxrtGearMarginHeight **Dimension** **2** **CSG**

Specifies the space on the top and bottom sides between the Toolbar's edge and its children.

XmNxrtGearMarginWidth **Dimension** **2** **CSG**

Specifies the space on the left and right sides between the Toolbar's edge and its children.

XmNxrtGearOrientation **unsigned char** **XmHORIZONTAL** **CSG**

Specifies whether the toolbar is to be displayed as a row (`XmHORIZONTAL`) or as a column (`XmVERTICAL`).

5.7.1 Toolbar Constraint Resources

XmNxrtGearBottomSpace **Dimension** **0** **CSG**

Specifies the space between the widget and the item below it.

XmNxrtGearLeftSpace **Dimension** **0** **CSG**

Specifies the space between the widget and the item above it.

XmNxrtGearPosition **int** **XmLAST_POSITION** **CSG**

Specifies the 0-based position of this widget within the list. Changing this value will cause it to change its visual position if it is managed. Setting a value of 0 or `XmLAST_POSITION` will move the widget to the beginning or end of the list.

XmNxrtGearRightSpace **Dimension** **0** **CSG**

Specifies the space between the widget and its next neighbor.

XmNxrtGearTopSpace **Dimension** **0** **CSG**

Specifies the space between the widget and the previous row.

5.7.2 Resources Inherited by Toolbar

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNinsertPosition	Composite
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNuserData	XmManager
XmNwidth	Core
XmNx	Core
XmNy	Core

5.8 Procedures and Methods Reference

This section lists the Toolbar procedures and methods in alphabetical order.

XmCreateXrtToolbar()
Creates a Toolbar widget.

```
Widget
XmCreateXrtToolbar(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmIsXrtToolbar()

Returns `True` if a widget is a Toolbar.

```
Boolean
XmIsXrtToolbar(
    Widget  widget
    )
```

XrtGearMrmInitialize()

Registers the XRT/gear widgets with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
void
XrtGearMrmInitialize(void)
```

XrtGearSetXmStringConverter()

Sets the XmRXmString resource converter to create XmString values.

```
void
XrtGearSetXmStringConverter(void)
```

The conversion uses the following syntax to specify embedded font changes (tags):

@font@	font's tag in `XmNfontList`
@P@	previous font
\@	a literal "@"

For example,

```
For a @bold@free evaluation@P@ of any product
```

with the fontlist:

```
fixed,-*-times-bold-r-normal-*-140-*=bold
```

displays:

For a **free evaluation** of any product

6

The Aligner Widget

Introduction ■ *Sample Program*
Widget Synopsis ■ *Margin and Spacing*
Aligner and XmSeparator ■ *Label Orientation*
Constraint Resources ■ *Resource Reference*
Procedures and Methods Reference

6.1 Introduction

Aligner is a Manager widget that provides a simple way to lay out a vertically arranged group of control widgets, each with an associated label (or other widget) placed to its left. The text of a control widget can be aligned with the baseline of its label.

Figure 33 illustrates the anatomy of a typical Aligner widget.

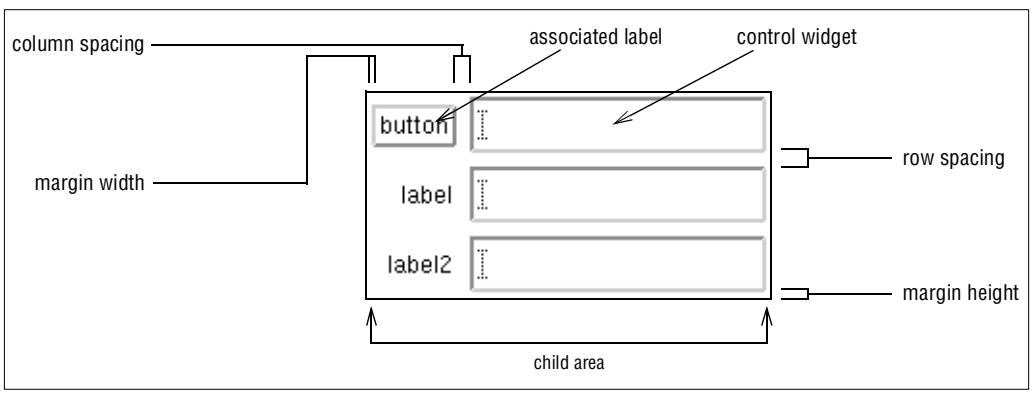


Figure 33 Anatomy of an Aligner widget

6.2 Sample Program

The following code creates an Aligner widget and its children:

```
form = XtVaCreateWidget("form",
    xmFormWidgetClass,          parent,
    NULL);
aligner = XtVaCreateManagedWidget("aligner",
    xmXrtAlignerWidgetClass,    form,
    NULL);
XtVaCreateManagedWidget("button",
    xmPushButtonWidgetClass,    aligner,
    NULL);
XtVaCreateManagedWidget("text",
    xmTextWidgetClass,          aligner,
    NULL);
XtVaCreateManagedWidget("label",
    xmLabelWidgetClass,         aligner,
    NULL);
XtVaCreateManagedWidget("text2",
    xmTextWidgetClass,          aligner,
    NULL);
XtVaCreateManagedWidget("label2",
    xmLabelWidgetClass,         aligner,
    NULL);
XtVaCreateManagedWidget("text3",
    xmTextWidgetClass,          aligner,
    NULL);
XtManageChild(form);
```

This code creates the following:

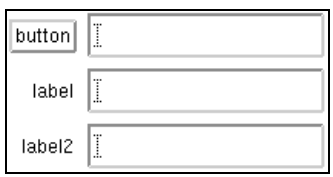


Figure 34 A sample Aligner

The children are laid out in the order they are created. Each label can be positioned to the left of or above its associated control widget. The controls can be configured to stretch horizontally or vertically when the Aligner widget is resized.

6.3 Widget Synopsis

Include File:	*<Xm/XrtAligner.h>*
Class Name:	XmXrtAligner
Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtManager → XmXrtAligner
Class Pointer:	xmXrtAlignerWidgetClass

The Aligner widget is subclassed from the XmXrtManager widget, which is in turn subclassed from the XmManager widget.

For more information on the Motif widget hierarchy and where Aligner fits into this hierarchy, see section 1.2 on page 8.

6.4 Margin and Spacing

You can specify the space between each label and its control, the space surrounding the child area, and the space between rows.

The `XmNxrtGearColumnSpacing` resource specifies the number of pixels between each label and its control. The default is 5.

The `XmNxrtGearMarginHeight` resource specifies the space above and below the child area. The `XmNxrtGearMarginWidth` resource specifies the space to the left and right of the child area. The default for both of these resources is 2.

The `XmNxrtGearRowSpacing` resource specifies the space, in pixels, between rows (label/control pairs). The default is 5.

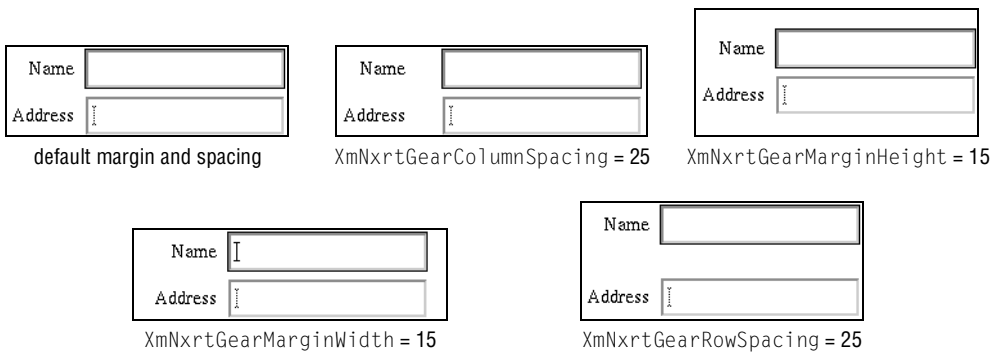


Figure 35 Aligner margin and spacing resources

6.5 Aligner and XmSeparator

When an XmSeparator widget is defined as a child of an Aligner widget, it is treated as its own row (in other words, it is not assumed to be part of a label/control pair). This allows you to divide your Aligner widget into sections for increased legibility.

For example, here is code that creates two label/control pairs and separates them:

```
form = XtVaCreateWidget("form",
    xmFormWidgetClass, parent,
    NULL);
aligner = XtVaCreateManagedWidget("aligner",
    xmXrtAlignerWidgetClass, form,
    NULL);
XtVaCreateManagedWidget("button",
    xmPushButtonWidgetClass, aligner,
    NULL);
XtVaCreateManagedWidget("text",
    xmTextWidgetClass, aligner,
    NULL);
XtVaCreateManagedWidget("separator",
    xmSeparatorWidgetClass, aligner,
    NULL);
XtVaCreateManagedWidget("label",
    xmLabelWidgetClass, aligner,
    NULL);
XtVaCreateManagedWidget("text2",
    xmTextWidgetClass, aligner,
    NULL);
```

This produces the following:

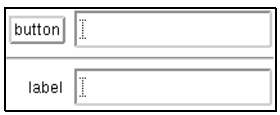


Figure 36 An Aligner containing a Separator

6.6 Label Orientation

The `XmNxrtGearOrientation` resource specifies the relative position of a label and its control widget. It can be set to either `XmHORIZONTAL` (label is to left of control) or `XmVERTICAL` (label is above control); the default is `XmHORIZONTAL`.[1]

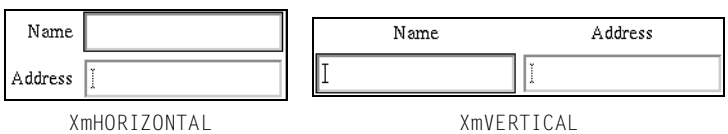


Figure 37 Aligner label orientation

1. When an Aligner widget is oriented vertically, `XmNxrtGearColumnSpacing` still specifies the space between each label and its control, and `XmNxrtGearRowSpacing` still specifies the distance between label/control pairs.

6.7 Constraint Resources

Aligner's constraint resources allow you to position a label relative to its control, provide resizing constraints, and specify the position of a child.

Baseline Alignment

XmNxrtGearAlignBaseline specifies whether the baseline of the label should be aligned with the baseline of its control; its default is True.

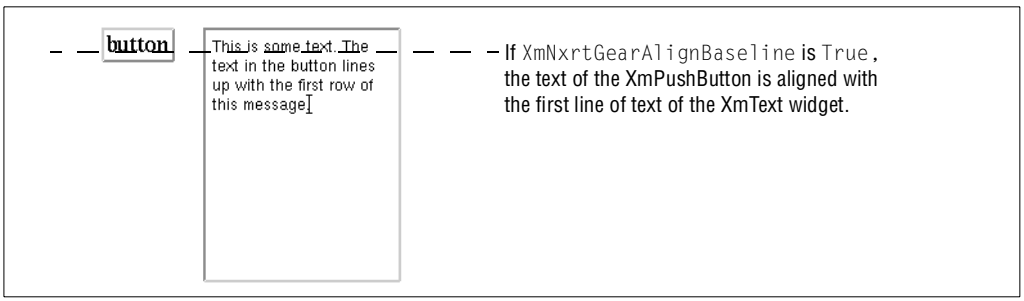


Figure 38 Aligner baseline alignment

XmNxrtGearAlignBaseline is only useful when the label and its control are subclassed from the XmLabel or XmText widget. XmNxrtGearAlignBaseline, if True, overrides XmNxrtGearAlignerAlignment.

Label Positioning

XmNxrtGearAlignerAlignment specifies the position of a label relative to its control if XmNxrtGearAlignBaseline is not set to True. XmNxrtGearAlignerAlignment can be set to the following:

```
XRTGEAR_ALIGNER_TOPRIGHT
XRTGEAR_ALIGNER_MIDDLERIGHT
XRTGEAR_ALIGNER_BOTTOMRIGHT
XRTGEAR_ALIGNER_TOPCENTER
XRTGEAR_ALIGNER_MIDDLECENTER
XRTGEAR_ALIGNER_BOTTOMCENTER
XRTGEAR_ALIGNER_TOPLEFT
XRTGEAR_ALIGNER_MIDDLELEFT
XRTGEAR_ALIGNER_BOTTOMLEFT
```

Topleft	Topcenter	Topright
Middleleft	Middlecenter	Middleright
Bottomleft	Bottomcenter	Bottomright

The above are for both XmVERTICAL and XmHORIZONTAL orientations.

Resize Control

XmNxrtGearResizeHeight and XmNxrtGearResizeWidth specify whether the control child should resize vertically and horizontally, respectively, when its row or column is resized. By default, both resources are False, and the control child is not resized.

Child Widget Position

XmNxrtGearPosition specifies the position of the child widget in the list of children. When the child widgets are created, they are numbered in order starting from 0. For

example, consider the following code, adapted from the example at the start of this chapter:

```
button = XtVaCreateManagedWidget("button",
    xmPushButtonWidgetClass,    aligner,
    NULL);
text1 = XtVaCreateManagedWidget("text",
    xmTextWidgetClass,          aligner,
    NULL);
label1 = XtVaCreateManagedWidget("label",
    xmLabelWidgetClass,         aligner,
    NULL);
text2 = XtVaCreateManagedWidget("text2",
    xmTextWidgetClass,          aligner,
    NULL);
label2 = XtVaCreateManagedWidget("label2",
    xmLabelWidgetClass,         aligner,
    NULL);
text3 = XtVaCreateManagedWidget("text3",
    xmTextWidgetClass,          aligner,
    NULL);
```

These child widgets are numbered from 0 to 5, as shown in Figure 39.

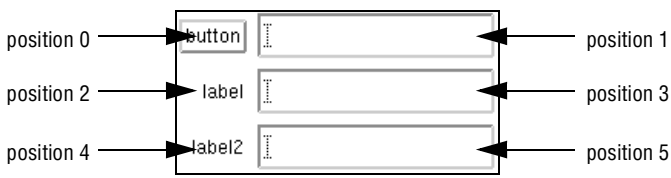


Figure 39 Aligner child widget positions

Changing the value of `XmNxrtGearPosition` changes the widget's visual position if the widget is managed. Whenever `XmNxrtGearPosition` is changed for one child widget, the `XmNxrtGearPosition` values for other child widgets are bumped up if necessary. For example, suppose the second text field has its position changed:

```
XtVaSetValues(text2,
    XmNxrtGearPosition, 2,
    NULL);
```

This change produces the following:

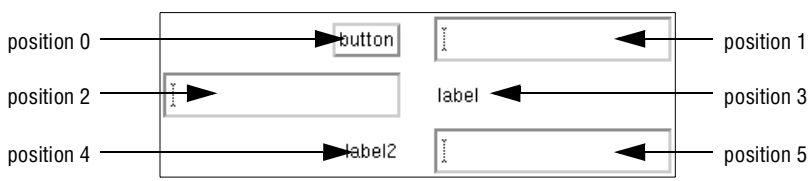


Figure 40 Aligner child widget positions after repositioning

When the second text widget is moved to position 2, the widget already in position 2 is bumped to position 3.

Setting XmNxrtGearPosition to 0 or XmLAST_POSITION moves the widget to the beginning or the end of the list.

6.8 Resource Reference

This section lists all of the resources for the Aligner widget. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

XmNxrtGearColumnSpacing **Dimension** **5** **CSG**

Specifies the space between each label and its control in pixels.

XmNxrtGearMarginHeight **Dimension** **2** **CSG**

Specifies the space above and below the child area in pixels.

XmNxrtGearMarginWidth **Dimension** **2** **CSG**

Specifies the space to the left and right of the child area in pixels.

XmNxrtGearOrientation **unsigned char** **XmHORIZONTAL** **CSG**

Specifies the relative position of each label to its control: XmHORIZONTAL (label is to the left of control), or XmVERTICAL (label is above control).

XmNxrtGearRowSpacing **Dimension** **5** **CSG**

Specifies the space between rows in pixels.

6.8.1 Aligner Constraint Resources

XmNxrtGearAlignerAlignment **XrtGearAlignerAlignment** **XRTGEAR_ALIGNER_CENTER** **CSG**

Specifies the position of a label relative to its control. The following are for both XmHORIZONTAL and XmVERTICAL:

XRTGEAR_ALIGNER_TOPRIGHT	top right
XRTGEAR_ALIGNER_MIDDLERIGHT	middle right
XRTGEAR_ALIGNER_BOTTOMRIGHT	bottom right
XRTGEAR_ALIGNER_TOPCENTER	top center
XRTGEAR_ALIGNER_MIDDLECENTER	middle center
XRTGEAR_ALIGNER_BOTTOMCENTER	bottom center
XRTGEAR_ALIGNER_TOPLEFT	top left
XRTGEAR_ALIGNER_MIDDLELEFT	middle left

XRTGEAR_ALIGNER_BOTTOMLEFT bottom left

XmNxrtGearAlignBaseline, if True, overrides XmNxrtGearAlignerAlignment.

XmNxrtGearAlignBaseline **Boolean** **True** **CSG**

Specifies whether the label's baseline should be aligned with the baseline of its control. This resource is ignored if the label and control are not subclassed from XmLabel, XmText or XmXrtGearCombo.

XmNxrtGearAlignBaseline is only useful when the label and its control are subclassed from the XmLabel, XmText or XmXrtGearCombo widget. XmNxrtGearAlignBaseline, if True, overrides XmNxrtGearAlignerAlignment.

XmNxrtGearPosition **int** **XmLAST_POSITION** **CSG**

Specifies the 0-based position of this widget within the list. Changing this value will cause it to change its visual position if it is managed. Setting a value of 0 or XmLAST_POSITION will move the widget to the beginning or end of the list.

XmNxrtGearResizeHeight **Boolean** **False** **CSG**

Specifies whether the control should resize vertically when its row changes height. If several controls set this resource, they will be stretched vertically in proportion to their preferred sizes.

XmNxrtGearResizeWidth **Boolean** **False** **CSG**

Specifies whether the control child should resize horizontally when its column changes width.

6.8.2 Resources Inherited by Aligner

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNinsertPosition	Composite
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNuserData	XmManager
XmNwidth	Core
XmNx	Core
XmNy	Core

6.9 Procedures and Methods Reference

This section lists the Aligner procedures and methods in alphabetical order.

XmCreateXrtAligner()
Creates an Aligner widget.

```
Widget
XmCreateXrtAligner(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmIsXrtAligner()

Returns `True` if a widget is an Aligner.

```
Boolean
XmIsXrtAligner(
    Widget  widget
    )
```

XrtGearMrmInitialize()

Registers the XRT/gear widgets with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
void
XrtGearMrmInitialize(void)
```

XrtGearSetXmStringConverter()

Sets the XmRXmString resource converter to create XmString values.

```
void
XrtGearSetXmStringConverter(void)
```

The conversion uses the following syntax to specify embedded font changes (tags):

@font@	font's tag in `XmNfontList`
@P@	previous font
\@	a literal "@"

For example,

```
For a @bold@free evaluation@P@ of any product
```

with the fontlist:

```
fixed,-*-times-bold-r-normal-*-140-*=bold
```

displays:

For a **free evaluation** of any product

7

The Outliner Widget

Getting Started ■ *Concepts*
Widget and Xt Object Synopses ■ *Creating a Tree*
Traversing the Tree ■ *Restructuring the Tree*
Controlling Tree Display ■ *Multiple Columns*
Customizing Nodes ■ *User Interaction*
Printing Outliner Widgets ■ *Performance Issues*
Resource Reference ■ *Procedures and Methods Reference*
Translations and Actions

7.1 Getting Started

The Outliner widget provides an easy way to display hierarchically-organized data. The data items can be organized into columns for easier display. Figure 41 shows examples of the kinds of data that can be displayed using the Outliner.

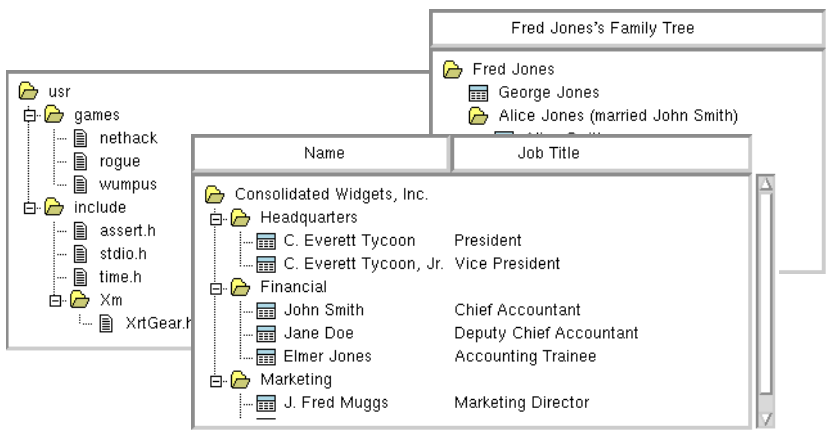


Figure 41 Example Outliner applications

7.1.1 Data Organization and Display

The data displayed by the Outliner widget is assumed to be in the form of a tree. Each item of data is either a *folder* (a branch of the tree) or an *item* (a leaf of the tree).

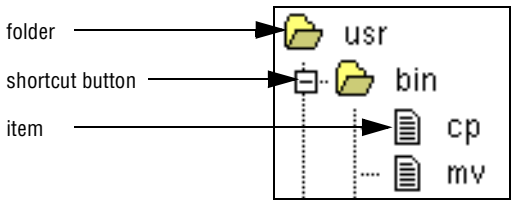


Figure 42 Data displayed in a tree

The end-user can control whether the data contained in a folder is to be displayed or not; if the data is not displayed, the folder is *collapsed*. The end-user can collapse or un-collapse a folder by either double-clicking on the folder or single-clicking on the *shortcut button*.

The shortcut button contains a character that indicates whether a folder is collapsed. If the button contains a + character, the folder is collapsed. If it contains a – character, the data contained in a folder is displayed.

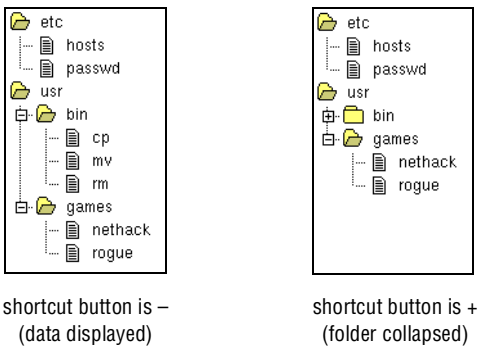


Figure 43 Shortcut button behavior

7.1.2 Other User Interaction

The Outliner widget also supports the following end-user interaction capabilities:

- Selecting nodes and folders
- Resizing columns
- Opening and closing folder nodes
- Traversing the tree
- Dragging and dropping nodes
- Dragging text from other widgets and dropping into the Outliner

For more information on user interaction, see section 7.10 on page 135.

7.1.3 Building a Tree Using the Outliner Widget

The easiest way to build a tree is to read it in from a data file. The following program creates a tree from a data file:

```
#include <stdio.h>
#include <stdlib.h>
#include <Xm/XrtGear.h>
#include <Xm/XrtOutliner.h>
#include <Xm/XrtNode.h>
#include <Xm/XrtNodeFolder.h>
#include <X11/Intrinsic.h>

int
main(int argc, char **argv)
{
    Widget          toplevel, outliner;
    XtAppContext    app_context;
    FILE            *infile;

    toplevel = XtAppInitialize(&app_context, "Sample", NULL, 0,
        &argc, argv, NULL, NULL, 0);
    outliner = XtVaCreateManagedWidget("outliner",
        xmXrtOutlinerWidgetClass, toplevel,
        NULL);
    infile = fopen("file.dat", "r");
    XrtGearNodeCreateTreeFromStream(outliner, infile, False,
        XRTGEAR_FOLDERSTATE_OPEN_ALL, NULL, NULL, NULL, NULL);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app_context);
}
```

In this case, data files consist of one line of text per data item, with tab characters used to specify the hierarchy levels. The following is an example of a data file (with *<tab>* representing the tab character):

```
etc
<tab>hosts
<tab>passwd
usr
<tab>bin
<tab><tab>cp
<tab><tab>mv
<tab><tab>rm
<tab>games
<tab><tab>nethack
<tab><tab>rogue
```

The following is the Outliner created from this data file:

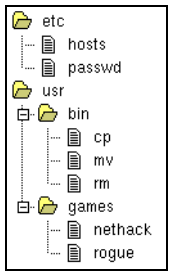


Figure 44 An Outliner widget created from a data file

The Outliner widget also makes it easy to create a tree from a data file containing a list of pathnames. For example, suppose you want to create a tree from the following file:

```
etc/hosts
etc/passwd
usr/bin/cp
usr/bin/mv
usr/bin/rm
usr/games/nethack
usr/games/rogue
```

To do this, change the call to *XrtGearNodeCreateTreeFromStream()* as follows:

```
XrtGearNodeCreateTreeFromStream(outliner, infile, True,
    XRTGEAR_FOLDERSTATE_OPEN_ALL, "/", NULL, NULL, NULL);
```

Setting the third parameter passed to *XrtGearNodeCreateTreeFromStream()* to `True` indicates that a complete path hierarchy is included with every line of data.The fifth parameter specifies the indent character to use, which in this case is the slash character `/`.

The tree created by this statement is identical to that created by the previous program, and is shown in Figure 44.

7.1.4 Proceeding From Here

At this point, you know everything you need to display simple hierarchical data using the Outliner widget. The following sections describe additional features and capabilities of Outliner, including multi-column display, tree sorting and traversal, and customization of node styles.

7.2 Concepts

In order to be able to use the advanced features of the Outliner widget, you will need to understand the following basic concepts:

- How to build trees using the Outliner widget, the NodeFolder object, and the Node object
- How to use the XrtString object to store text strings or pixmaps
- How to use the XrtList object to store lists of elements

7.2.1 The Outliner Widget and Trees

The Outliner widget is a Manager widget that displays hierarchical information in the form of a tree. Each item of data in the tree is called a *node*; the node data itself consists of one or more text strings called *node labels*. The area in which nodes are displayed is called the *node area*.

Nodes are of two types: *items* and *folder nodes*. The only difference between these two types of nodes is that a folder node can have one or more children. XRT/gear defines two Xt Objects which represent nodes: the Node object, which represents an item, and the NodeFolder object, which represents a folder node. The NodeFolder object is subclassed from the Node object, which means that NodeFolders can use resources defined for Nodes.

The Outliner widget itself is considered the root node of the tree, and can be treated as if it were a NodeFolder object; however, it is not displayed.

Figure 45 illustrates these widgets and objects, and the tree concept. Note that the Outliner widget, despite being a Manager widget, cannot have children that are not items or folder nodes or columns.

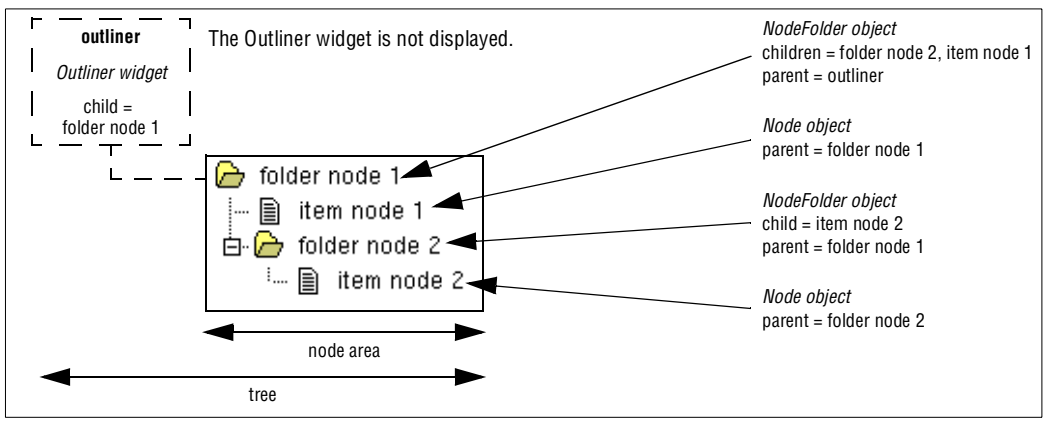


Figure 45 Tree and node terminology

7.2.2 The XrtString Object

The XrtString object stores data as a String, an XmString, or an XrtGearIcon pixmap.

To create an XrtString object that stores a String, call the *XrtGearStringCreateCharString()* procedure:

```
XrtGearObject   mystring;

mystring = XrtGearStringCreateCharString("my string");
```

In the Outliner widget, XrtStrings are used by the `XmNxrtGearLabel` resource to specify a single label for nodes. See section 7.8.1 on page 118 for more information on node labels.

To create an XrtString object that stores an XmString, call *XrtGearStringCreateXmString()*:

```
XrtGearObject   mystring;
XmString        xmstring;

xmstring = XmStringCreate("my string", XmFONTLIST_DEFAULT_TAG);
mystring = XrtGearStringCreateXmString(xmstring);
XmStringFree(xmstring);
```

To create an XrtString object that stores an XrtGearIcon pixmap, call *XrtGearStringCreateIconString()*.

XRT/gear provides methods and procedures that manipulate XrtString objects. See section 13.2 on page 297 for a description of these methods and procedures.

7.2.3 XrtList Objects

An XrtList object is a convenient way to store a list of elements of the same size. To create an XrtList object, call *XrtGearListCreate()*. To add elements to a list, call *XrtGearListAppend()*. The following is an example of an XrtList object that uses these methods:

```
XrtGearObject   list, label;
Widget          newnode;
Widget          folder;
...
list = XrtGearListCreate(sizeof(XrtGearObject));
label = XrtGearStringCreateCharString("label1");
XrtGearListAppend(list, (XtPointer)&label);
label = XrtGearStringCreateCharString("label2");
XrtGearListAppend(list, (XtPointer)&label);
label = XrtGearStringCreateCharString("label3");
XrtGearListAppend(list, (XtPointer)&label);
```

This creates a list of XrtString objects and appends three elements to the list. The following code retrieves the first object in the list:

```
myxrtstr = *(XrtGearObject *)XrtGearListGetItem(list, 0);
```

When used as the value of the `XmNxrtGearLabel` resource, the XrtList object enables you to specify multiple labels for the same node. In this case, the XrtList object must

be a list of XrtStrings. See section 7.8.1 on page 118 for more information on node labels.

XRT/gear provides several methods and procedures that manipulate XrtList objects, including:

XrtGearListDelete()	delete a list element
XrtGearListGetItemCount()	get the number of elements in the list
XrtGearListSort()	sort the list

For a complete description of the XrtList procedures and methods, see section 13.1 on page 289.

7.3 Widget and Xt Object Synopses

Outliner widget	Include Files:	*<Xm/XrtOutliner.h>*
	Class Name:	XmXrtOutliner
	Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtOutliner
	Class Pointer:	xmXrtOutlinerWidgetClass
Node object	Include Files:	*<Xm/XrtNode.h>*
	Class Name:	XmXrtNode
	Class Hierarchy:	Object → XmXrtNode
	Class Pointer:	xmXrtNodeObjectClass
NodeFolder object	Include Files:	*<Xm/XrtNodeFolder.h>*
	Class Name:	XmXrtNodeFolder
	Class Hierarchy:	Object → XmXrtNode → XmXrtNodeFolder
	Class Pointer:	xmXrtNodeFolderObjectClass
NodeStyle object	Include Files:	*<Xm/XrtNodeStyle.h>*
	Class Name:	XmXrtNodeStyle
	Class Hierarchy:	Object → XmXrtNodeStyle
	Class Pointer:	xmXrtNodeStyleObjectClass
Column object	Include Files:	*<Xm/XrtColumn.h>*
	Class Name:	XmXrtColumn
	Class Hierarchy:	Object → XmXrtColumn
	Class Pointer:	xmXrtColumnObjectClass

7.4 Creating a Tree

XRT/gear provides three ways to create a tree:

- From a tree data file
- From a string buffer
- Directly, using calls to *XtVaCreateWidget()* and *XtCreateWidget()*

7.4.1 Data File Format

Each line of a tree data file defines one node of the tree, and tab characters at the start of each line supply the tree hierarchy. The following is an example of a tree data file (with <*tab*> representing a tab character):

```
item1
folder
<tab>item2
<tab>item3
<tab>subfolder
<tab><tab>item4
item5
```

Here, `item1`, `folder` and `item5` are children of the root Outliner widget; their input lines contain no tab characters. `item2`, `item3` and `subfolder` are children of `folder`, and `item4` is a child of `subfolder`.

By default, nodes without children are assumed to be items. To create a folder without children, append the string `(F)` to the node name:

```
item1
folder1(F)
item2
```

This mechanism also supports C, O, I, A to specify folder state, where:

```
C   XRTGEAR_FOLDERSTATE_CLOSED
O   XRTGEAR_FOLDERSTATE_OPEN_FOLDERS
I   XRTGEAR_FOLDERSTATE_OPEN_ITEMS
A   XRTGEAR_FOLDERSTATE_OPEN_ALL
```

For example:

```
folder1(F,C)
```

Data File Special Characters

In a tree data file, the `(` character (open parenthesis) is a special character called the *open escape character*. When a `(` character is detected, anything following the `(` is treated as a special instruction. This ends when a `)` character is detected; the `)` character is known as the *close escape character*. The comma character is also a special character; it is used to separate multiple labels for a node.

To include a special character in a node name, precede its left and right parenthesis with backslashes:

```
\(bracketed name\)
```

This defines a node named `(bracketed name)`.

To include a backslash in a node name, use two backslashes. Backslashes preceding non-special characters are ignored.

7.4.2 Creating a Tree From a Data File

To create a tree from a data file, call the *XrtGearNodeCreateTreeFromStream()* method:

```
infile = fopen("file2.datFILE2.DAT", "r");
XrtGearNodeCreateTreeFromStream(outliner, infile, False,
    XRTGEAR_FOLDERSTATE_OPEN_ALL, "-", NULL, "{", "}");
```

XrtGearNodeCreateTreeFromStream() requires eight parameters:

- The first parameter points to the parent Outliner widget.
- The second parameter specifies the data file, and points to a previously opened file descriptor.
- The third parameter specifies whether the complete hierarchy is supplied with each node.
- The fourth parameter specifies the initial folder state for the folders in the tree. (Folder states are described in 7.10.3 on page 139.)
- The fifth parameter specifies the indent (tree hierarchy) character used in the file. If this parameter is `NULL`, the tab character is assumed.
- The sixth parameter specifies the character used to separate multiple labels for a node. Only the first label will be displayed, unless multiple columns are defined. Columns are discussed in section 7.8 on page 118. If this parameter is `NULL`, the comma character is assumed.
- The seventh parameter specifies the open escape character used in the file; if `NULL`, the `{` character is assumed.
- The eighth parameter specifies the close escape character used in the file; if `NULL`, the `}` character is assumed.

If the third parameter to *XrtGearNodeCreateTreeFromStream()* is `True`, the complete hierarchy is supplied with each node, as in the following example data file:

```
users/dave/bin
users/dave/lib
users/keith/bin
users/keith/lib
users/fang
```

Here, if the indent string (the fifth parameter) is set to the slash character, the following tree is produced by *XrtGearNodeCreateTreeFromStream()*:

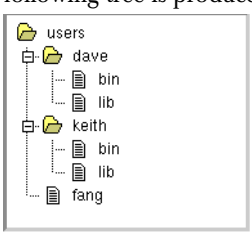


Figure 46 An Outliner created from a data file

Note that folder nodes are created when necessary. For example, when the line

```
users/dave/bin
```

is read, the folder nodes `users` and `dave` are created, along with the item node bin. These folder nodes are re-used when

```
users/dave/lib
```

is read.

If the third parameter to *XrtGearNodeCreateTreeFromStream()* is `False`, the hierarchy is inferred from the indent levels in the file. For example, the following data file produces the tree shown in Figure 46 when the seventh parameter is `NULL` (assuming the tab character is used as the indent string):

```
users
<tab>dave
<tab><tab>bin
<tab><tab>lib
<tab>keith
<tab><tab>bin
<tab><tab>lib
<tab>fang
```

Creating Subtrees

XrtGearNodeCreateTreeFromStream() can also be used to create subtrees of existing trees. To do this, pass a node of the existing tree as the first parameter:

```
XrtGearNodeCreateTreeFromStream(node, infile, False,
    XRTGEAR_FOLDERSTATE_OPEN_ALL, NULL, NULL, NULL, NULL);
```

Here, the new subtree is created as a child of the node pointed to by *node*.

If anything other than an Outliner widget or NodeFolder object is passed to *XrtGearNodeCreateTreeFromStream()*, a container "root" folder is created and positioned between the passed widget and the tree specified in the file.

Specifying a Tree Data File in a Resource File

To specify a tree data file and its associated special characters from a resource file or a builder tool, set the `XmNxrtGearTreeData` resource, as in the following example:

```
*.xrtGearTreeData:  test.dat FOLDERSTATE_OPEN_ALL - DEFAULT { }
```

The format of the tree data file is identical to that passed to *XrtGearNodeCreateTreeFromStream()*.

The resource string for `XmNxrtGearTreeData` consists of one or more syntax elements, separated by one or more spaces. The following syntax elements are supported, in the order given:

filename folderstate separator indent open_escape closed_escape

Only the first element, *filename*, is required; this is the name of the tree data file. (In this example, the tree data file name is `test.dat`.) (If `XmNxrtGearTreeData` is set by a call to *XtSetValues()* or from a builder tool and *filename* is `NULL`, any existing nodes will be destroyed.)

The subsequent syntax elements perform the following functions:

- *indent* specifies the indent (tree hierarchy) character used in the file. If this is set to `DEFAULT`, the tab character is assumed.
- *separator* specifies the character used to separate multiple labels for a node. Only the first label will be displayed, unless multiple columns are defined. If this is set to `DEFAULT`, the comma character is assumed.
- *open_escape* specifies the open escape character used in the file; if set to `DEFAULT`, the `{` character is assumed.
- *closed_escape* specifies the close escape character used in the file; if set to `DEFAULT`, the `}` character is assumed.

Each of these syntax elements is optional; however, if you want to specify an element, you must specify all of its preceding elements. (For example, if you want to supply *indent*, you must also supply *pathtype* and *separator*. These can be set to `DEFAULT` if you do not want to change them.)

You can also use `XmNxrtGearTreeData` in your program:

```
XtVaSetValues(outliner,
    XmNxrtGearTreeData,     "test.dat - DEFAULT { }",
    NULL);
```

The value of this resource is a string whose format is the same as that used in resource files.

7.4.3 Creating a Tree From a Character String

The Outliner widget enables you to create a tree from the contents of a character string. To do this, call *XrtGearNodeCreateTreeFromBuffer()*:

```
static char data[] =
    "item1\n\
    folder\n\
    \titem2\n\
    \titem3\n\
    \tsubfolder\n\
    \t\titem4\n\
    item5";

XrtGearNodeCreateTreeFromBuffer(outliner, data, False,
    XRTGEAR_FOLDERSTATE_OPEN_ALL, "-", NULL, "{", "}");
```

The second parameter to *XrtGearNodeCreateTreeFromBuffer()* is the character string containing the tree data; it is assumed to be a multi-line character string in the same format as that expected by *XrtGearNodeCreateTreeFromStream()*. All other parameters are identical to their equivalents in *XrtGearNodeCreateTreeFromStream()*.

7.4.4 Manually Creating a Tree

You can also create a tree yourself by calling *XtVaCreateManagedWidget()* to create the parent Outliner widget and then repeatedly calling *XtVaCreateWidget()* to create the nodes of the tree. The following is an example of a simple tree created in this fashion:

```
outliner = XtVaCreateManagedWidget("outliner",
    xmXrtOutlinerWidgetClass, toplevel,
    NULL);
folder = XtVaCreateWidget("folder",
    xmXrtNodeFolderObjectClass, outliner,
    XmNxrtGearLabel, XrtGearStringCreateCharString("Folder"),
    NULL);
item1 = XtVaCreateWidget("item",
    xmXrtNodeObjectClass, outliner,
    XmNxrtGearLabel, XrtGearStringCreateCharString("Item1"),
    NULL);
item2 = XtVaCreateWidget("item2",
    xmXrtNodeObjectClass, folder,
    XmNxrtGearLabel, XrtGearStringCreateCharString("Item2"),
    NULL);
```

This creates the following tree:

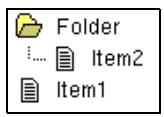


Figure 47 A manually-created tree

The `XmNxrtGearLabel` resource specifies the text displayed by a NodeFolder or Node object. In this example, this text is of type XrtString, and specifies a single label.

7.4.5 Preserving the Tree

The Outliner widget provides two means of preserving the contents of trees you have created:

- In a data file
- In a buffer (of type String)

In both cases, the tree is written out in the data file format described in section 7.4.1 on page 98.

Writing a Tree to a File

To preserve a tree in a file, call *XrtGearNodeWriteTreeToStream()*:

```
Widget      outliner;
FILE        *outfile;
int         length;
Boolean     ok;

ok = XrtGearNodeWriteTreeToStream(outliner, outfile, False,
    False, "/", NULL, NULL, NULL, &length);
```

The parameters passed to this method are identical to those passed to *XrtGearNodeCreateTreeFromStream()*, with the following exceptions:

- The second parameter specifies an open file into which the tree is to be written.
- The fourth parameter is a Boolean value specifying whether the parent node is to be included as part of the tree to be written; `True` indicates that the parent node is to be included. (If this parameter is `True` and the parent node is the Outliner widget, the name of the Outliner widget, as returned by *XtName()*, is written to the file.)
- An optional final parameter returns the length of the written file in bytes.

This method returns `True` if the write operation was successful, or `False` if an error occurred.

Writing a Tree to a Buffer

To preserve a tree in a buffer, call *XrtGearNodeWriteTreeToBuffer()*. This method writes the tree as a text string in the Outliner data file format.

```
Widget      outliner;
char        buffer[256];
int         length;

XrtGearNodeWriteTreeToBuffer(outliner, buffer, False,
    False, "/", NULL, NULL, NULL, &length);
```

This method is identical to *XrtGearNodeWriteTreeToStream()*, except for the following:

- This method returns a pointer to internal memory in which the text string is stored. (This memory can be overwritten at any time.)
- The second parameter, if non-`NULL`, specifies a buffer into which the text string is written. Supplying your own buffer ensures that your text string will not be overwritten later in the program.
- The final parameter, if non-`NULL`, returns the length of the buffer in bytes.

7.5 Traversing the Tree

XRT/gear provides several resources and methods which allow you to traverse trees once they have been created. This enables you to find particular labels in a tree, or retrieve a node for later relocation. (Tree restructuring is described in section 7.6 on page 107.)

7.5.1 Getting the List of Node Children

To retrieve the list of children of a particular node, or of the Outliner widget itself, get the value of its XmNxrtGearNodeChildList resource. You can then retrieve a particular child from the list, as shown in the following example:

```
XrtGearObject   list;
Widget          folder, firstchildnode;

XtVaGetValues(folder,
    XmNxrtGearNodeChildList,    &list,
    NULL);
firstchildnode = *(Widget *)XrtGearListGetItem(list, 0);
```

The XmNxrtGearNodeChildList resource cannot be set.

The returned value of XmNxrtGearNodeChildList is an XrtList object that lists the children of the node in the order in which they are displayed. For more information on XrtList objects, see section 7.2.3 on page 96.

7.5.2 The Current Node

The tree node that currently has focus is called the *current node*. You can determine the current node by getting the value of the Outliner widget's XmNxrtGearNodeCurrent resource.

Normally, the current node is specified by the end-user. To set the current node yourself, either set XmNxrtGearNodeCurrent to the desired current node, or call the *XrtGearNodeTraverseTo()* method. If there is no current node specified, XmNxrtGearNodeCurrent is NULL.

Node Change Callbacks

You can supply callbacks to be called whenever the current node changes as the result of a user action or a call to *XrtGearNodeTraverseTo()*. (Changing XmNxrtGearNodeCurrent does not trigger these callbacks.) To do this, add these callbacks to the list specified by the XmNxrtGearNodeCurrentChangedCallback resource.

These callbacks are passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          container;      /* read-only */
    Widget          previous_node;  /* read-only */
    Widget          new_node;
    Boolean         doit;
} XrtGearNodeCurrentChangedCallbackStruct;
```

The *previous_node* and *new_node* elements specify the previous current node and new current node, respectively. *container* is the parent Outliner widget. To disable a current node change, set *doit* to False when *reason* is XRTGEAR_REASON_NODE_CHANGE_BEGIN.

The following is an example of a NodeCurrentChanged callback. This callback prints the number of labels defined for the new current node.

```
void
changeCB(Widget outliner, XtPointer client_data,
XrtGearNodeCurrentChangedCallbackStruct *call_data)
{
    XrtGearObject       label;

    if (call_data->reason != XRTGEAR_REASON_NODE_CHANGE_BEGIN) {
        return;
    }
    if (call_data->new_node == NULL) {
        return;
    }
    XtVaGetValues(call_data->new_node,
        XmNxrtGearLabel, &label,
        NULL);
    if (XrtGearListIsList(label)) {
        fprintf (stderr, "node has %d labels\n",
            XrtGearListGetItemCount(label));
    } else {
        fprintf (stderr, "node has one label\n");
    }
}
```

7.5.3 Traversal Convenience Methods

XRT/gear provides the following convenience methods for tree traversal:

XrtGearNodeGetChild(node, n)	Returns the *n*th child of a node.
XrtGearNodeGetFirstInTree(root)	Returns the first child of the tree root.
XrtGearNodeGetFirstNonCollapsedInTree(root)	Returns the first non-collapsed child of the tree root. (A node is *collapsed* if it is a descendant of a folder node that is closed. For more information on collapsed nodes, see section 7.10 on page 135.)
XrtGearNodeGetLastInTree(root)	Returns the last child in a hierarchical list of nodes.
XrtGearNodeGetLastNonCollapsedInTree(root)	Returns the last non-collapsed child in a hierarchical list of nodes.
XrtGearNodeGetLevel(root, node)	Returns the level (or generation) of the given node as compared to its root node.
XrtGearNodeGetNextInTree(root, node)	Given a node, returns the node following it in the hierarchical list.

XrtGearNodeGetNextNonCollapsedInTree(root, node)	Given a node, returns the non-collapsed node following it in the hierarchical list.
XrtGearNodeGetNumChildren(node)	Returns the number of children of a widget or object (as specified by the `XmNxrtGearNodeChildList` resource).
XrtGearNodeGetPreviousInTree(root, node)	Given a node, returns the node preceding it in the hierarchical list.
XrtGearNodeGetPreviousNonCollapsedInTree(root, node)	Given a node, returns the non-collapsed node preceding it in the hierarchical list.
XrtGearNodeGetVisibility(node)	Returns the current visibility status of a node.
XrtGearNodeGetWidgetParent(node)	Travels up the node hierarchy until a widget (normally the Outliner) is reached, and returns it.
XrtGearNodeIsAncestor(root, ancestor, node)	Returns `True` if one specified node is an ancestor of another.
XrtGearNodeIsInCollapsedBranch(root, node)	Returns `True` if the specified node is in a collapsed branch relative to the provided root.

Figure 48 illustrates the relationship between a hierarchical list of nodes and some of the convenience methods listed above.

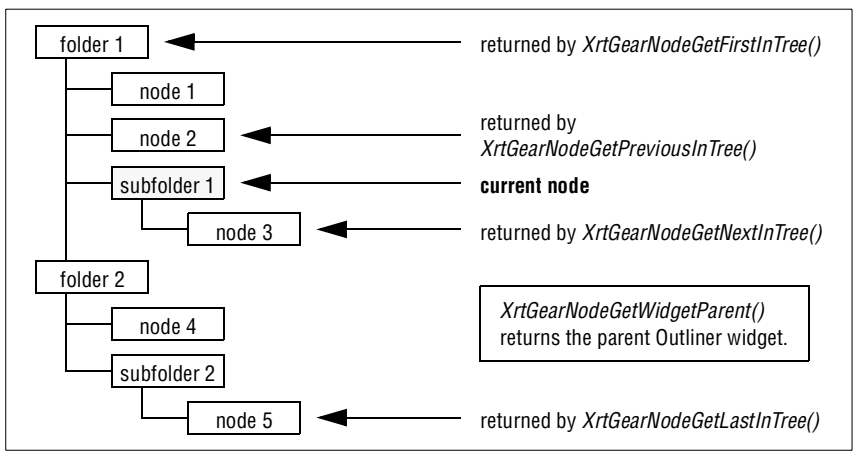


Figure 48 Behavior of hierarchical list convenience methods, when passed the current node

For a more complete description of the convenience methods discussed in this section, see section 7.13.2 on page 170.

7.6 Restructuring the Tree

XRT/gear enables you to restructure a tree by moving nodes from one location to another or by sorting the nodes in the tree.

7.6.1 Moving Nodes

Nodes can be rearranged in a tree by calling *XrtGearNodeSetParent()*. This method defines a new parent for a node or subtree:

```
XrtGearNodeSetParent(subtree, newparent, sibling);
```

The new parent can be a NodeFolder object, or the Outliner widget (If the new parent is `NULL`, the subtree is cut). If the new parent has other children, the node or subtree is added immediately preceding the node specified by *sibling*. Figure 49 illustrates the effect of *XrtGearNodeSetParent()*.

If *sibling* is not specified, the node or subtree is added to the end of the list of children. If `XmNxrtGearAutoSort` (described in section 7.6.3 on page 108) is set to `True`, *sibling* is ignored, and the child is inserted in its sorted position.

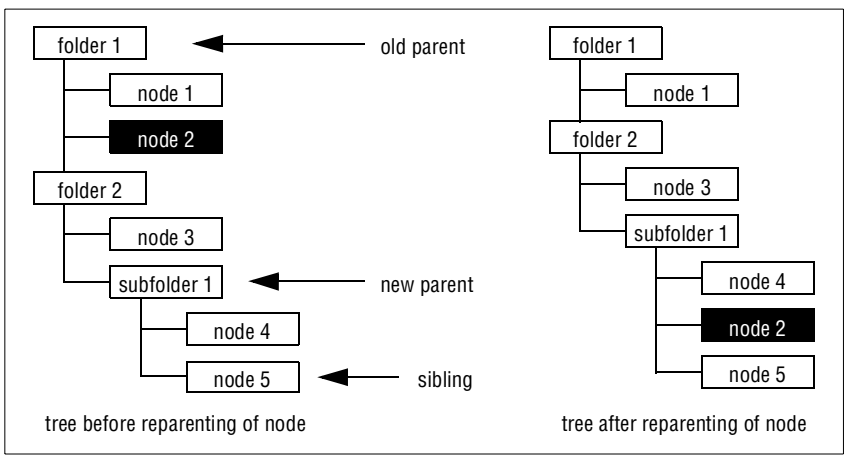


Figure 49 Reparenting a node

Movement Callbacks

You can specify move callbacks to be invoked whenever *XrtGearNodeSetParent()* is called, or whenever the end-user moves or copies a node or subtree. Move callbacks enable you to restrict or monitor tree movement.

To register a move callback, add it to the list specified by the XmNxrtGearNodeMoveCallback resource. These callbacks are passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    unsigned char   operation;      /* read-only */
    Widget          node;           /* read-only */
    Widget          old_parent;     /* read-only */
    Widget          new_parent;     /* read-only */
    Widget          new_next;
    Boolean         doit;
} XrtGearNodeMoveCallbackStruct;
```

The *operation* member is either XmDROP_MOVE (operation is a move) or XmDROP_COPY (operation is a copy).

If *XrtGearNodeSetParent()* is called, *old_parent* contains the old parent of the moved node, and *new_parent* contains the new one. *new_next* contains the new next sibling.

new_next and *doit* are modifiable when *reason* is XRTGEAR_REASON_NODE_MOVE_BEGIN. To abort a move operation, set *doit* to False . To change the position of *node* in the list of children of *new_parent*, change *new_next*, which specifies the sibling that is to immediately follow *node* in the child list.

7.6.2 Deleting Nodes

To delete a node, call *XtDestroyWidget(node)* or *XrtGearNodeDestroy*().

7.6.3 Sorting Nodes

The Outliner widget's XmNxrtGearAutoSort resource, when set to True, enables you to automatically sort nodes against their siblings as they are created. The sort procedure set by XmNxrtGearNodeCompareProc is used if it is defined.

By default, XmNxrtGearAutoSort is False, and sorting is not automatically performed.

To manually sort a tree or a subtree, call *XrtGearNodeSortTree()*:

```
XrtGearNodeSortTree(tree);
```

By default, this sorts each group of siblings in alphabetical order. An example of this is shown in Figure 50.

If a node contains multiple labels, the labels are treated as if they are one single concatenated string for sorting purposes. (To sort the labels within a node, use *XrtGearListSort()*.)

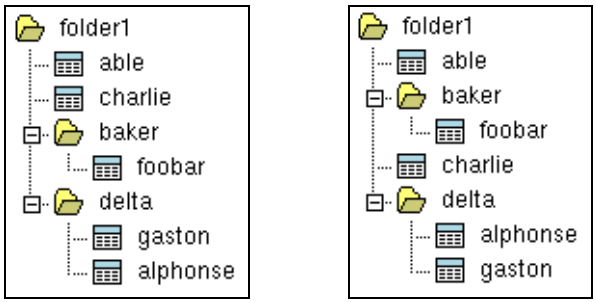


Figure 50 Sorting a tree

Note that the move callbacks specified by `XmNxrtGearNodeMoveCallback` are not invoked when the tree is sorted.

Specifying the Sort Criterion

By default, sorting is in alphabetical order. To change how node sorting is performed, define a node comparison procedure and set the value of the `XmNxrtGearNodeCompareProc` resource to point to this procedure.

The node comparison procedure must accept two parameters of type XtPointer, which are pointers to two nodes to be compared. It must return an integer as follows:

- A positive integer if the first item is to be considered "greater than" the second
- Zero if the two items are equivalent
- A negative integer if the first item is to be considered "less than" the second

For example, the following sets node sorting to be in reverse alphabetical order:

```
int
my_node_compare_proc(XtPointer item1, XtPointer item2)
{
    char        buffer1[1024], buffer2[1024];
    Widget      node1 = *(Widget *)item1;
    Widget      node2 = *(Widget *)item2;

    assert(XmIsXrtNode(node1));
    assert(XmIsXrtNode(node2));
    XrtGearNodeCvtLabelToString(node1, ",", buffer1);
    XrtGearNodeCvtLabelToString(node2, ",", buffer2);
    return (strcmp(buffer2, buffer1));
}
...
XtVaSetValues(outliner,
    XmNxrtGearNodeCompareProc,  my_node_compare_proc,
    NULL);
```

7.7 Controlling Tree Display

The Outliner widget enables you to control the following tree display attributes:

- Height and width sizing policy
- Margins
- Colors and fonts
- Node indent and vertical node spacing
- Visibility
- Display of individual nodes
- Focus highlight
- Resizing

The appearance of individual nodes can also be controlled by defining node styles, which are described in section 7.9 on page 129.

7.7.1 Height and Width

The Outliner widget uses three concepts when defining widget height and width:

- The *actual* height and width of the widget's window, as displayed on the screen.
- The *virtual* height and width, which are the height and width needed to display the whole widget.
- The *preferred* height and width, which are the height and width desired by the application.

Actual Height and Width

The actual height and width of the Outliner widget are specified by the `XmNheight` and `XmNwidth` resources. This height and width include the height and width of any column labels or scrollbars included as part of the widget, as shown in Figure 51. (Column labels are discussed in section 7.8.5 on page 123; scrollbars are described in section 7.10.6 on page 141.)

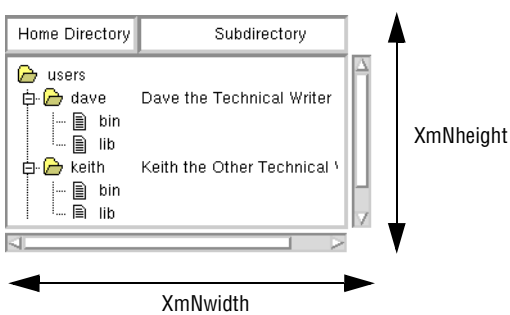


Figure 51 XmNheight, XmNwidth, and the Outliner widget

Virtual Height and Width

The virtual height and width of the Outliner widget are specified by the XmNxrtGearHeightVirtual and XmNxrtGearWidthVirtual resources. These resources represent what the height and width of the widget would be if it were entirely displayed on the screen, and are read-only.

Virtual height and width are normally only used by the horizontal and vertical scrollbars if they are present. See section 7.10.6 on page 141 for a discussion of scrollbars.

Preferred Height

The preferred height of the Outliner widget is calculated to be the following:

- The height of the column labels, if any (see section 7.8.5 on page 123 for a discussion of column labels), plus
- The height of the horizontal scrollbar, if managed (see section 7.10.6 on page 141 for a discussion of scrollbars), plus
- The preferred height of the node area.

The preferred height of the node area is, in turn, calculated from the values of the XmNxrtGearHeightSizingPolicy and XmNxrtGearHeightPreferredCount resources, as follows:

- If XmNxrtGearHeightSizingPolicy is set to XRTGEAR_HEIGHT_PIXEL, the preferred height of the node area is taken from the value of XmNxrtGearHeightPreferredCount.
- If XmNxrtGearHeightSizingPolicy is set to XRTGEAR_SIZE_CHAR, the height of the highest font in the list specified by XmNfontList is multiplied by the value of XmNxrtGearHeightPreferredCount to yield the preferred height of the node area.
- If XmNxrtGearHeightSizingPolicy is set to XRTGEAR_SIZE_NODE, the preferred height of the node area is calculated as follows:
  - The number of nodes, specified by XmNxrtGearHeightPreferredCount, is multiplied by the value of XmNxrtGearNodeHeight.
  - The following is then added:

    ```
    (XmNxrtGearHeightPreferredCount - 1) * XmNxrtGearNodeSpacing
    ```

    (XmNxrtGearNodeSpacing is discussed in section 7.7.4 on page 113.)
  - Finally, twice the value of XmNhighlightThickness (discussed in section 7.7.8 on page 115) is added.

  Basically, this is the size of the nodes to be displayed, added to the size of the space between the nodes, added to the space needed to display the focus highlight.
- If XmNxrtGearHeightSizingPolicy is set to XRTGEAR_SIZE_VARIABLE, the heights of all the non-collapsed nodes are added together to produce the preferred height of the node area. (A node is *collapsed* if it is a descendant of a folder node that is closed.)

Whenever possible, the Outliner widget is sized to its preferred height when displayed. If this is not possible, the height of the node area is reduced accordingly. If the height of the widget is greater than the preferred height, the node area is stretched accordingly.

Preferred Width

The preferred width of the Outliner widget is calculated to be the following:

- The width of the vertical scrollbar, if managed (see section 7.10.6 on page 141 for a discussion of scrollbars), plus
- The preferred width of the node area.

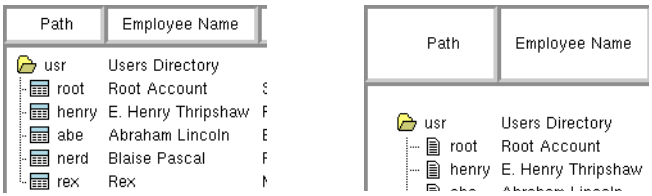
The preferred width of the node area is, in turn, calculated from the values of the `XmNxrtGearWidthSizingPolicy` and `XmNxrtGearWidthPreferredCount` resources, as follows:

- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_PIXEL`, the preferred width of the node area is taken from the value of `XmNxrtGearWidthPreferredCount`.
- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_CHAR`, the unit of measurement is the character specified by `XmNxrtGearWidthChar`, in the first font in the list specified by `XmNfontList`. This unit of measurement is multiplied by the value of `XmNxrtGearWidthPreferredCount` to yield the preferred width of the node area. (By default, `XmNxrtGearWidthChar` is the character `M`.)
- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_COLUMN`, the average width of the columns is multiplied by the value of `XmNxrtGearWidthPreferredCount` to yield the preferred width of the node area. (By default, an Outliner widget has only one column. Multi-column widgets are discussed in section 7.8 on page 118.)
- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_VARIABLE`, the widths of all the non-collapsed nodes are added together to produce the preferred width of the node area.
- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_MAXIMUM`, the widths of all nodes are added together to produce the preferred width of the node area.

Whenever possible, the Outliner widget is sized to its preferred width when displayed. If this is not possible, the width of the node area is reduced accordingly. If the width of the widget is greater than the preferred width, the node area is stretched accordingly. (If a multi-column display is defined, the last column is stretched. See section 7.8 on page 118 for a description of multi-column displays.)

7.7.2 Margins

The `XmNxrtGearMarginHeight` and `XmNxrtGearMarginWidth` resources control the distance in pixels between the widget and its border, and between column labels and their borders, if defined. Figure 52 illustrates the behavior of these resources.



margin height and width = 5 (default)

margin height and width = 20

Figure 52 Margin height and width

7.7.3 Colors and Fonts

The foreground colors and the fonts used in the Outliner widget are controlled by the XrtColumn and XrtNodeStyle objects. For information on setting the column label foreground color and font, see section 7.8.7 on page 124. For information on node styles and setting node foreground colors and fonts, see section 7.9 on page 129.

To change the background color of the Outliner widget, set the standard Motif `XmNbackground` resource.

7.7.4 Spacing

You can control the amount of space to indent each level of nodes in the display of an Outliner widget, as well as the vertical space between nodes.

Node Level Indent

The `XmNxrtGearNodeIndent` resource controls the amount of indenting space (in pixels) to use to distinguish a folder node from its children. Its default value is 20 pixels. The effects of this resource are shown in Figure 53.



`XmNxrtGearNodeIndent` = 10 pixels

`XmNxrtGearNodeIndent` = 50 pixels

Figure 53 Node indent

Vertical Node Spacing

The Outliner widget's `XmNxrtGearNodeSpacing` resource specifies the space between the top and bottom of any two consecutive nodes. This value must be greater than or equal to the value of `XmNhighlightThickness` to ensure that there is always enough room to draw the focus box.

By default, XmNxrtGearNodeSpacing is twice the value of XmNhighlightThickness.

7.7.5 Visibility

A node is said to be visible if it is currently being displayed in the Outliner's node area. The Outliner widget provides several features which control node visibility. You can:

- Retrieve the list of visible nodes
- Set the topmost visible node
- Make a node visible
- Obtain the current visibility status of a node

Retrieving the List of Visible Nodes

To retrieve the current list of visible nodes, get the value of the Outliner widget's XmNxrtGearNodeVisibleList resource. This list is an XrtList object. XrtList objects are described in section 7.2.3 on page 96.

Setting the Topmost Visible Node

To position a node as the topmost visible node in the node area, set the XmNxrtGearNodeTop resource to point to the desired Node or NodeFolder object:

```
Node        mytop;

XtVaSetValues(outliner,
    XmNxrtGearNodeTop,          mytop,
    NULL);
```

XmNxrtGearNodeTop also enables you to quickly retrieve the first node in the visible node list (which is faster than getting the value of XmNxrtGearNodeVisibleList and examining its first element).

Making a Node Visible

To make a tree node visible, call *XrtGearNodeMakeVisible()*:

```
Node        node;

XrtGearNodeMakeVisible(node);
```

If the node is partially or fully obscured, the tree will be repositioned to fully display the specified node.

Retrieving the Visibility Status

To get the current visibility status of a node, call *XrtGearNodeGetVisibility()*:

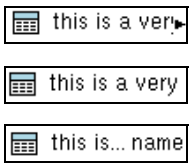
```
Node        node;

XrtGearNodeGetVisibility(node);
```

The returned XmVisibility value is one of XmVISIBILITY_UNOBSCURED, XmVISIBILITY_PARTIALLY_OBSCURED or XmVISIBILITY_FULLY_OBSCURED.

7.7.6 Text Clipping

The Outliner's XmNxrtGearTextHorizClipPolicy resource specifies how text should be clipped if it is too large to fit in the available space. Figure 54 illustrates the valid values of this resource, and the resulting clipping behaviors when given the string "this is a very long name":



XRTGEAR_TRUNCATE_WITH_ARROW

XRTGEAR_TRUNCATE

XRTGEAR_COMPRESS_WITH_ELLIPSIS

Figure 54 Text clipping behaviors

The default value is XRTGEAR_TRUNCATE_WITH_ARROW.

7.7.7 Node Height

The height of each node in a tree is specified by the Outliner widget's XmNxrtGearNodeHeight resource. By default, the height of each node is the height of the tallest node style.

7.7.8 Focus Highlight

When a node becomes the current node, a focus highlight is drawn around the node. The size of this focus highlight in pixels is specified by the XmNhighlightThickness resource. The default value of this resource is 2 pixels.

Figure 55 illustrates the focus highlight.

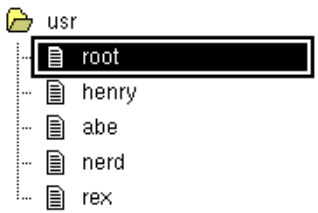


Figure 55 The focus highlight

The focus highlight is drawn in two colors. The outer half of the focus highlight is drawn in the color specified by the XmNhighlightColor resource, and the inner half is drawn in the background color (specified by XmNbackground). If the value of XmNhighlightThickness is odd, the outer half will be one pixel thicker than the inner half.

The following code sets the highlight thickness and color:

```
outliner = XtVaCreateManagedWidget("outliner",
    xmXrtOutlinerWidgetClass,       toplevel,
    XmNhighlightThickness,          4,
    XmNxrtGearNodeSpacing,          4,
    XtVaTypedArg, XmNhighlightColor,
        XmRString, "blue", strlen("blue")+1,
    NULL);
```

Note that if the value of `XmNhighlightThickness` is greater than the value of `XmNxrtGearNodeSpacing`, the focus highlight will overlap the neighboring nodes.

7.7.9 Resizing

In the Outliner widget, the height and width resizing behavior are controlled by the `XmNxrtGearHeightResizePolicy` and `XmNxrtGearWidthResizePolicy` resources. These resources accept the following values:

`XRTGEAR_WIDGET_RESIZE_VARIABLE`	Resize to the preferred height determined by `XmNxrtGearHeightSizingPolicy` or `XmNxrtGearWidthSizingPolicy`, if allowed by the parent. (See section 7.7.1 on page 110 for more details on these resources.) This is the default.
`XRTGEAR_WIDGET_RESIZE_GROW_ONLY`	Resize only if the resize operation increases the size of the widget (if the parent permits it); otherwise, leave unchanged.
`XRTGEAR_WIDGET_RESIZE_FIXED`	Don't resize unless a parent requests it.

Resize Callbacks

You can specify callbacks to be called whenever the Outliner widget is resized. To do this, add the callback name to the widget's `XmNxrtGearResizeCallback` resource.

Each resize callback is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    Xevent          *event;         /* read-only */
    Dimension       width;
    Dimension       height;
} XrtGearResizeCallbackStruct;
```

width and *height* are the new width and height, and can be modified.

Here is an example of a very simple resize callback:

```
void
resizeCB(Widget outliner, XtPointer client_data,
XrtGearResizeCallbackStruct *call_data)
{
    XrtGearDisplayPolicy    hpolicy, vpolicy;

    if (call_data->width < 200) {
        vpolicy = XRTGEAR_DISPLAY_NEVER;
    } else {
        vpolicy = XRTGEAR_DISPLAY_AS_NEEDED;
    }
    if (call_data->height < 200) {
        hpolicy = XRTGEAR_DISPLAY_NEVER;
    } else {
        hpolicy = XRTGEAR_DISPLAY_AS_NEEDED;
    }
    XtVaSetValues(outliner,
        XmNxrtGearScrollBarHorizDisplayPolicy, hpolicy,
        NULL);
    XtVaSetValues(outliner,
        XmNxrtGearScrollBarVertDisplayPolicy, vpolicy,
        NULL);
}
```

This callback makes the appropriate scrollbar disappear if the widget is shrunk below 200 pixels. (For more information on scrollbars, see section 7.10.6 on page 141.)

7.7.10 Cursor Display

The Outliner widget's `XmNxrtGearCursor` resource controls the appearance of the cursor over the displayed widget. The value of this resource is of type Cursor, and the resource's default value is the arrow cursor .

The `XmNxrtGearCursorResizeWidth` resource specifies the cursor used when resizing the width of a column. The value of this resource is also of type Cursor, and the resource's default value is the column resize pointer .

Tracking Cursor Movement

You can program the Outliner widget to change the appearance of the cursor each time the mouse is moved over the edge of a node, or moved inside or outside a node. To do this:

- Define a cursor tracking callback, and add it to the callback list defined by the `XmNxrtGearCursorTrackingCallback` resource.
- Set the `XmNxrtGearCursorTracking` resource to `True` to indicate that cursor tracking is to begin.

Each callback in the list specified by `XmNxrtGearCursorTrackingCallback` is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          object;         /* read-only */
    Cursor          old_cursor;     /* read-only */
    Cursor          new_cursor;
} XrtGearCursorTrackingCallbackStruct;
```

reason indicates the mouse movement that triggered the callback, and is one of the following:

`XRTGEAR_REASON_CURSOR_EDGE_LEFT`	cursor is moving over left edge
`XRTGEAR_REASON_CURSOR_EDGE_RIGHT`	cursor is moving over right edge
`XRTGEAR_REASON_CURSOR_INSIDE`	cursor is moving inside node or label
`XRTGEAR_REASON_CURSOR_LEAVE`	cursor is leaving node or label

In each case, *new_cursor* is the new look of the cursor. You can change the cursor appearance by changing this value in your callback procedure.

7.8 Multiple Columns

The Outliner widget enables you to define multiple labels for any node of your tree. Normally, only the first label for each node is displayed when the tree is displayed.

To display multiple labels for your tree, you can define additional columns to be associated with the parent Outliner widget. Each column displays one additional label for each node of your tree.

7.8.1 Defining Multiple Labels

The text displayed by a NodeFolder or Node object is specified by the `XmNxrtGearLabel` resource. When this resource is set, its value can be either an XrtString object or an XrtList object. An XrtString object is used when a single label is defined for a node, and an XrtList object (containing a list of XrtStrings) is used when multiple labels are defined.

The following code creates an XrtList object and sets it as the value of a node's `XmNxrtGearLabel` resource:

```
XrtGearObject   list, label;
Widget          newnode;
Widget          folder;
...
list = XrtGearListCreate(sizeof(XrtGearObject));
label = XrtGearStringCreateCharString("label1");
XrtGearListAppend(list, (XtPointer)&label);
label = XrtGearStringCreateCharString("label2");
XrtGearListAppend(list, (XtPointer)&label);
label = XrtGearStringCreateCharString("label3");
XrtGearListAppend(list, (XtPointer)&label);

newnode = XtVaCreateWidget("newnode",
    xmXrtNodeObjectClass,       folder,
    XmNxrtGearLabel,            list,
    NULL);
```

This defines three labels for a node: *label1*, *label2*, and *label3*. When the tree is displayed, only the first label, *label1*, appears unless multiple columns have been defined.

Another approach is to use *XrtGearNodeCvtStringToLabel()*:

```
XrtGearObject   list, label;
Widget  node;
list = XrtGearNodeCvtStringToLabel("label1,label2,label3", ",");
```

This method requires three parameters: the node, the string to be converted into labels, and the separation character to be used to separate multiple labels. In this case, the labels *label1*, *label2,* and *label3* are defined for the node.

Note that `XmNxrtGearLabel` is automatically set for each node whenever *XrtGearNodeCreateTreeFromStream()* or *XrtGearNodeCreateTreeFromBuffer()* is called.

For more information on the XrtString and XrtList objects, see section 7.2 on page 95.

XrtList Objects and Data Files

When building a tree using *XrtGearNodeCreateTreeFromStream()* or *XrtGearNodeCreateTreeFromBuffer()*, you can specify multiple labels for a node in a data file by separating the labels with commas. For example:

```
item1
folder
<tab>item2
<tab>multilabel1,multilabel2,multilabel3
<tab>subfolder
<tab><tab>item4
item5
```

Here, the second child of `folder` is a node with three labels. An XrtList object is created for these labels, and becomes the value of the node's `XmNxrtGearLabel` resource.

To use a separation character other than a comma, set the sixth parameter passed to *XrtGearNodeCreateTreeFromStream()* or *XrtGearNodeCreateTreeFromBuffer()*. See section 7.4.2 on page 99 for more details on the parameters to these functions.

Labels and Resource Files

If the value of `XmNxrtGearLabel` is being specified in a resource file, all you need to do is supply the character string you want to use as the label:

```
*.node1.xrtGearLabel:   mystring
```

To specify multiple labels, use commas to separate them:

```
*.node1.xrtGearLabel:   multilabel1,multilabel2,multilabel3
```

7.8.2 Creating New Columns

XRT/gear defines a special object, the XrtColumn object, to represent a column. To create a column, create an XrtColumn object with the Outliner widget as its parent:

```
XtVaCreateWidget("column1", xmXrtColumnObjectClass, outliner, NULL);
```

This automatically causes a column to be displayed when the tree is displayed. Figure 56 shows an example of a multi-column tree display.

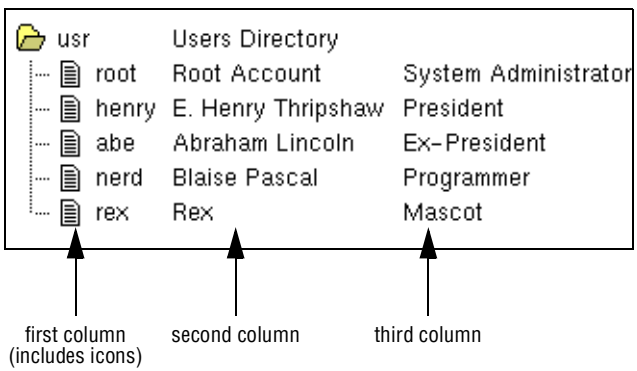


Figure 56 A multi-column tree display

The first created column displays the tree hierarchy and the first label for each node. Subsequent columns each display one additional node label for each node, if it exists. If more node labels exist than there are columns, the extra labels are not displayed.

The Outliner widget automatically creates a default column for you. If you create any columns yourself, the default column is destroyed. This means that if you create columns yourself, you must create all the columns you need. (If you destroy all the columns you have created, a new default column is automatically created.)

To obtain the list of columns defined for an Outliner widget, get the value of the `XmNxrtGearColumnList` resource. This list is an XrtList object whose elements are XrtColumn objects. `XmNxrtGearColumnList` is read-only, and cannot be created or set.

7.8.3 Ordering Columns

Normally, columns are displayed in the order in which they are created. To rearrange the order of column display, call the *XrtGearOutlinerMoveColumns()* method. This method moves a specified number of columns to a particular target destination.

The columns to be moved are either placed to the left of the destination column or to the right, depending on the following rules:

- If the columns to be moved are already immediately to the left of the destination column, they are placed to the right of the destination column.
- In all other cases, the moved columns are placed to the left of the destination column.

Figure 57 illustrates this behaviour.

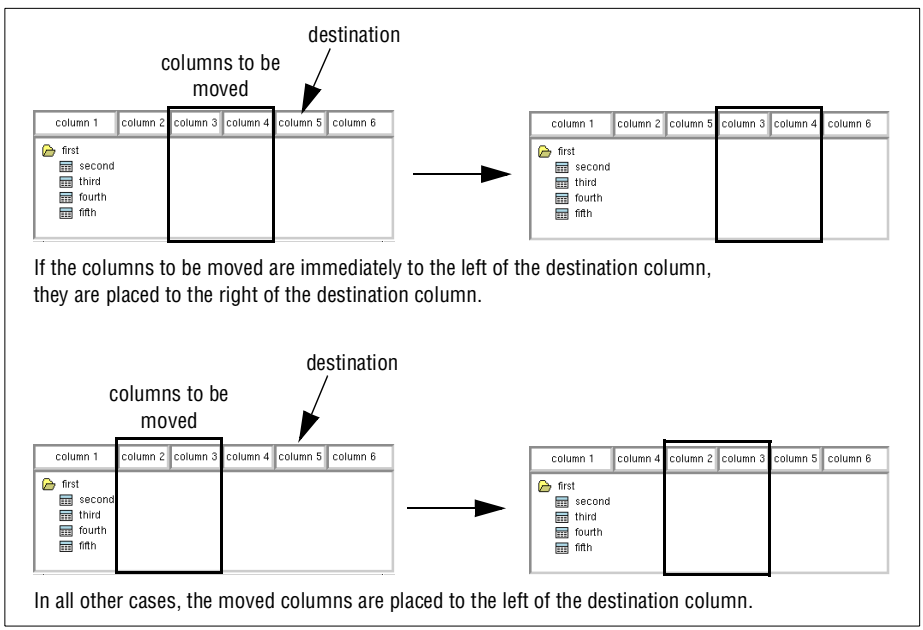


Figure 57 Column move behaviour

For example, the following code moves columns 5 and 6 in front of column 3:

```
int destination = 3;
int source = 5;
int num_to_move = 2;

XrtGearOutlinerMoveColumns(outliner, destination, source,
    num_to_move);
```

Note that the leftmost column, the tree hierarchy itself, has index 0; it cannot be moved and cannot serve as the destination or the source. (It must always appear first.)

Current Column Order

The current order in which columns are displayed is specified by the `XmNxrtGearColumnOrderList` resource. This resource returns an XrtList object whose elements are the XrtColumn objects in the order in which they are displayed. `XmNxrtGearColumnOrderList` is read-only, and cannot be created or set. (Columns can only be reordered by calling *XrtGearOutlinerMoveColumns()*.)

Note that the end-user can also rearrange the order in which columns appear. For details on column relocation, see section 7.10.10 on page 145.

7.8.4 Selected Columns

The column currently selected by the end-user can be obtained by getting the value of the `XmNxrtGearColumnSelected` resource:

```
Widget outliner, currcolumn;

XtVaGetValues(outliner,
    XmNxrtGearColumnSelected,   &currcolumn,
    NULL);
```

`XmNxrtGearColumnSelected` can be set, which changes the currently selected column.

For more information on selection and other user interactions, see section 7.10 on page 135.

Column Selection Callbacks

You can specify callbacks to be invoked whenever the end-user selects a column. To do this, add the callbacks to the list defined by the `XmNxrtGearColumnSelectionCallback` resource. This resource is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          old_column;     /* read-only */
    Widget          new_column;
    Boolean         doit;
} XrtGearColumnSelectionCallbackStruct;
```

old_column is the previous selected column, and *new_column* is the column that is about to be selected. From within selection callbacks, when *reason* is `XRTGEAR_REASON_SELECT_BEGIN` you can:

- Change the value of *new_column* to change the newly selected column.
- Set *doit* to `False` to disable the column selection operation.

7.8.5 Column Labels

You can define a label for each of the XrtColumn objects displayed in the tree. To create a label, set the XrtColumn object's `XmNxrtGearLabel` resource:

```
XtVaSetValues(column,
    XmNxrtGearLabel, XrtGearStringCreateCharString("col1label"),
    NULL);
```

The Outliner widget's `XmNxrtGearColumnLabelsShow` resource controls column label display. By default, this resource is set to `False`, and the labels are not displayed.

7.8.6 Column Indexing

Columns are indexed by their integer value. The following example can be used to get the column widget from its integer value:

```
XrtGearObject   column_list;
int             column_index;
Widget          column, outliner;

XtVaGetValues (outliner,
    XmNxrtGearColumnList,&column_list,
    NULL);

column = *( Widget *)XrtGearListGetItem(column_list, column_index);
```

The following example gets the index of a column widget:

```
XrtGearObject column_list;
int             column_index;
Widget          column, outliner;

XtVaGetValues(outliner,
    XmNxrtGearColumnList,&column_list,
    NULL);

column_index = XrtGearListFind(column_list, (XtPointer)column);
```

7.8.7 Controlling Column Display

The Outliner widget enables you to control the following column display attributes:

- Column positioning
- Column label foreground/background color
- Column label fonts
- Node and label text alignment
- Height and width
- Margin left and right
- Column and node area borders
- Resize behavior and callbacks

Column Label Foreground/background Color

The foreground/background color for each column label can be controlled by setting the XrtColumn object's `XmNforeground` (foreground color) and `XmNbackground` (background color) resources. By default, this color is inherited from the foreground/background color of the parent Outliner widget.

Column Label Fonts

The XrtColumn object provides resources which control the font to be used in column labels. The XrtColumn object's `XmNxrtGearFontListSelected` resource specifies the font to be used when the column is selected, and the `XmNfontList` resource specifies the font to be used in the label when the column is not selected. By default, these fonts are inherited from the parent Outliner widget's `XmNfontList` resource.

Node and Label Alignment

The alignment of the node text within a column is specified by the XrtColumn's `XmNxrtGearNodeAlignment` resource. The alignment of the text within a column label is specified by the XrtColumn's `XmNxrtGearLabelAlignment` resource. Both resources accept any of the following values:

`XRTGEAR_ALIGN_BEGINNING`	Align to start of text area
`XRTGEAR_ALIGN_CENTER`	Align to center of text area
`XRTGEAR_ALIGN_END`	Align to end of text area

Figure 58 shows the effects of changing column node and label alignment.

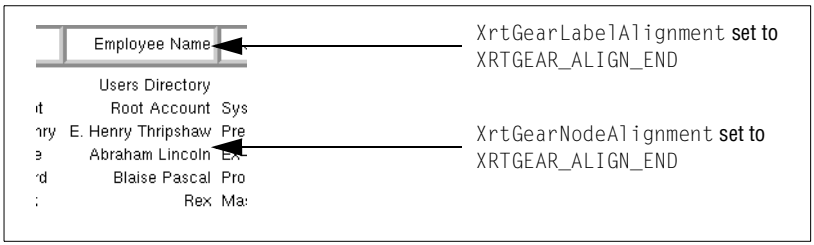


Figure 58 Column node and label alignment

The default value for `XmNxrtGearNodeAlignment` is `XRTGEAR_ALIGN_BEGINNING`. The default for `XmNxrtGearLabelAlignment` is `XRTGEAR_ALIGN_CENTER`.

Note that `XmNxrtGearNodeAlignment` does not affect the appearance of the first column (which contains the hierarchy information). This column is always left-aligned.

Column Height

The height of the column is specified by the XrtColumn object's `XmNheight` resource. This resource is automatically set to the height required to display the column's label, and does not normally need to be changed programmatically.

Column Width

The XrtColumn object uses two concepts when defining column width:

- The *actual* width of the column, as displayed on the screen.
- The *preferred* width, which is the width desired by the application.

The actual width of the column is specified by the `XmNwidth` resource. The preferred width of the column is calculated from the values of the XrtColumn's `XmNxrtGearWidthSizingPolicy` and `XmNxrtGearWidthPreferredCount` resources, as follows:

- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_PIXEL`, the preferred width of the column is taken from the value of `XmNwidth`.
- If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_CHAR`, the unit of measurement is the character specified by the parent Outliner widget's `XmNxrtGearWidthChar` resource, in the first font in the list specified by `XmNfontList`. This unit of measurement is multiplied by the value of

XmNxrtGearWidthPreferredCount to yield the preferred width of the column. (By default, XmNxrtGearWidthChar is the character M, and XmNxrtGearWidthPreferredCount is 10.)

- If XmNxrtGearWidthSizingPolicy is set to XRTGEAR_SIZE_LABEL_VARIABLE, the width of the column label is used as the preferred width of the column.
- If XmNxrtGearWidthSizingPolicy is set to XRTGEAR_SIZE_VARIABLE (the default value), the width of the widest label in this column is used as the preferred width of the column.

Whenever possible, the column is sized to its preferred width when displayed. If this is not possible, the width of the column is reduced accordingly. If the width of the widget is greater than the preferred width, the column is stretched accordingly.

If XmNwidth is 0, the column is hidden. Its borders are not displayed, and it cannot be interactively resized by the end-user.

Margin Left and Right

The distance in pixels between the left edge of the column label and the start of the column label text is specified by the XmNxrtGearMarginLeft resource. The distance in pixels between the end of the column label text and the right edge of the column label is specified by the XmNxrtGearMarginRight resource. The effects of these resources are shown in Figure 59.

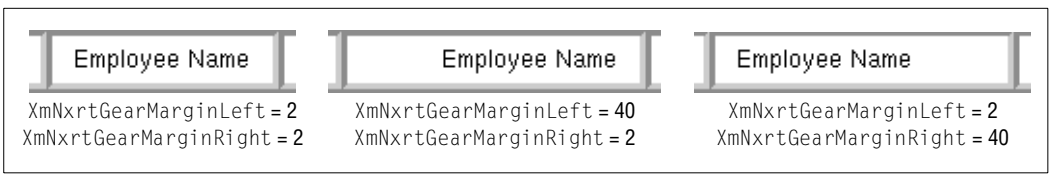


Figure 59 Column label margins

By default, both of these resources are set to 2.

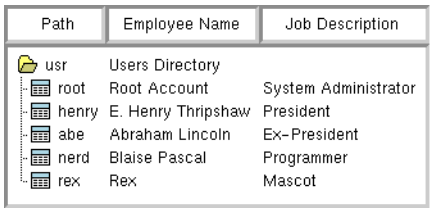
The top and bottom margins for the column label are controlled by the Outliner's XmNxrtGearMarginHeight and XmNxrtGearMarginWidth resources, which are discussed in section 7.7.2 on page 112.

Column and Node Area Borders

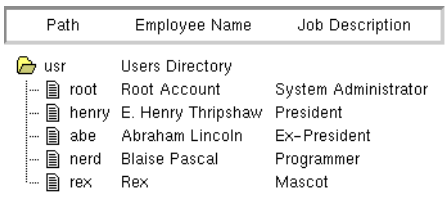
Column label and node area border drawing is specified by the XmNxrtGearBorderLocation resource. This resource enables you to specify that borders are to be drawn around the column label area, node label area, individual column labels, or any combination of the above. Valid values of XmNxrtGearBorderLocation and their effects are shown in Figure 60.



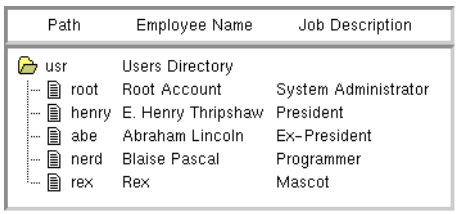
XRTGEAR_BORDER_LOC_EACH_LABEL



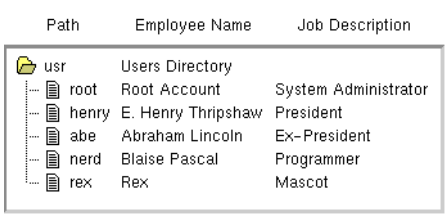
XRTGEAR_BORDER_LOC_EACH_LABEL_SEPARATED_FROM_NODES (default)



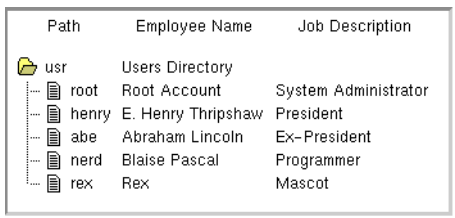
XRTGEAR_BORDER_LOC_LABELS



XRTGEAR_BORDER_LOC_LABELS_SEPARATED_FROM_NODES



XRTGEAR_BORDER_LOC_NODES



XRTGEAR_BORDER_LOC_LABELS_AND_NODES

Figure 60 Values of XmNxrtGearBorderLocation

The type of border drawn is specified by the Outliner's `XmNxrtGearBorderType` resource. Valid values are shown in Figure 61.

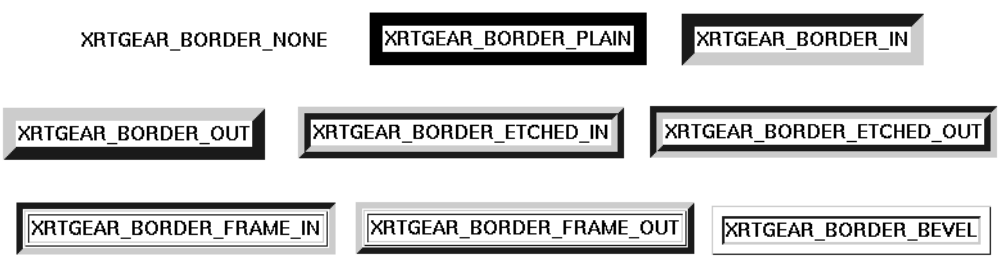


Figure 61 Outliner border types

The width of the borders drawn is specified by XmNxrtGearShadowThickness.

Interactive Column Resizing

Three XrtColumn resources control the end-user's column resizing behavior:

- XmNxrtGearResizePolicy specifies whether resizing is permitted. Its default is True, which permits resizing.
- XmNxrtGearWidthMinimum and XmNxrtGearWidthMaximum specify the minimum and maximum size, in pixels, to which the end-user can resize the column. Resizings performed by the program itself are not affected by the values of these resources. The default values for these resources are 10 and 10000, respectively.

You can specify callbacks to be invoked when a column is resized by adding to the callback list defined by the XmNxrtGearResizeCallback resource. Each callback in this list is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Dimension       width;
    Dimension       height;
    Dimension       old_width;      /* read-only */
    Dimension       old_height;     /* read-only */
    Boolean         doit;
} XrtGearEnhancedSizeCallbackStruct;
```

You can control column resizing as follows:

- You can modify the new height and width by changing the values of the *height* and *width* structure members. *old_width* and *old_height* contain the width and height before resizing.
- You can suppress the column resize by setting *doit* to False.

Also note that the end-user cannot resize a column if XmNxrtGearShadowThickness is 0, or if the column's XmNwidth resource is set to 0.

The changes listed above are only possible when *reason* is XRTGEAR_REASON_RESIZE_BEGIN.

Note that this callback structure is compatible with the XrtGearResizeCallbackStruct callback structure (used by the Outliner widget's XmNxrtGearResizeCallback resource).

7.9 Customizing Nodes

The Outliner widget provides a number of resources and convenience procedures that enable you to control the appearance of individual nodes being displayed. The following node behaviors can be modified, among others:

- The node label
- The background and foreground colors for selected and deselected nodes
- The font for selected and deselected nodes
- The folder icons in use
- The appearance of the lines connecting folders with their children

7.9.1 Node Label

The text for a node is defined by the Node object's `XmNxrtGearLabel` resource. The value of this resource can either be an XrtString object (a single label) or an XrtList object (multiple labels for the same node).

For more information on manipulating node labels, see section 7.8.1 on page 118.

Editing Node Labels

Each node label in every column can be edited individually. To allow validation by the application, `XmNxrtGearNodeValidateCallback` is used before and after an edit, allowing programmers to accept or disallow a change. To take this a step further, specific nodes and columns can be marked as editable or non-editable, overriding the node and folder settings. To go even further, the entire outline can be marked as editable or non-editable, overriding both node/column settings and the marking of any specific node or folder as editable/non-editable.

Even if a cell is marked as editable, `XmNxrtGearNodeEnterCellCallback` is used before and after a traversal so programmers can accept or disallow the mapping of the text widget to specific cells. Access to the text widgets is provided so programmers can customize things such as colors and fonts.

The widget used to edit the node uses the node's foreground and background colors, and font.

7.9.2 Node Styles

In the Outliner widget, most of the appearance behaviors of a node being displayed are controlled by the node's *node style*. The node style for a particular node is specified by a NodeStyle object. Node styles make it easier to create groups of nodes that look the same.

Creating a Node Style

To create a NodeStyle object, call *XmCreateXrtNodeStyle()*:

```
Widget      outliner, nodestyle;

nodestyle = XmCreateXrtNodeStyle(outliner, "mystyle", NULL, NULL);
```

The node style must be created as a child of the Outliner widget. It is destroyed when the Outliner widget is destroyed.

The Default Node Style

When an Outliner widget is created, a node style is automatically created and becomes the *default node style* for the tree. Each node of the tree automatically uses this node style unless the node's `XmNxrtGearNodeStyle` resource (described later in this section) is set.

To set the default node style yourself, set the Outliner's `XmNxrtGearNodeStyleDefault` resource to the node style you want to use as the default:

```
Widget      outliner, nodestyle;

XtVaSetValues(outliner,
    XmNxrtGearNodeStyleDefault, nodestyle,
    NULL);
```

The behavior of the pre-defined default node style is defined in the following section.

Node Style Resources

The following NodeStyle object resources control the appearance of a node:

`XmNxrtGearBackgroundSelected`	the background color of selected nodes
`XmNforeground`	the foreground color of deselected nodes
`XmNxrtGearForegroundSelected`	the foreground color of selected nodes
`XmNfontList`	the font for deselected nodes
`XmNxrtGearFontListSelected`	the font for selected nodes

Other NodeStyle resources control the appearance of folder icons (discussed in section 7.9.3 on page 131) and lines connecting folders to their children (discussed in section 7.9.4 on page 133).

The following code sets some of the above resources:

```
Widget outliner, nodestyle;

nodestyle = XmCreateXrtNodeStyle(outliner,
    "mystyle", NULL, NULL);
XtVaSetValues(nodestyle,
    XtVaTypedArg,               XmNxrtGearBackgroundSelected,
    XmRString, "white", strlen("white")+1,
    NULL);
XtVaSetValues(nodestyle,
    XtVaTypedArg,               XmNxrtGearFontListSelected,
    XmRString, "-*-terminal-bold-r-*-*-14-*-*-*-*_*-iso8859-*",
    strlen("-*-terminal-bold-r-*-*-14-*-*-*-*_*-iso8859-*")+1,
    NULL);
```

The pre-defined default node style uses the following values for the resources defined above:

- The values of `XmNbackground`, `XmNforeground` and `XmNfontList` are inherited from their parent widgets.
- The value of `XmNxrtGearSelectedBackground` is inherited from `XmNbackground`.
- The value of `XmNxrtGearSelectedForeground` is inherited from `XmNforeground`.
- The value of `XmNxrtGearFontListSelected` is inherited from `XmNfontList`.

Defining a Node Style for a Node

To associate a node style with a particular node, set the node's `XmNxrtGearNodeStyle` resource. For example, the following uses the node style named `mystyle`:

```
Widget outliner, node, nodestyle;

nodestyle = XrtGearNodeStyleFromName(outliner, "mystyle");
XtVaSetValues(node,
    XmNxrtGearNodeStyle,        nodestyle,
    NULL);
```

The *XrtGearNodeStyleFromName()* convenience method, shown above, retrieves a NodeStyle object when given its name.

The master list of node styles available to an application is defined by the Outliner widget's `XmNxrtGearNodeStyleList` resource:

```
XrtGearObject stylelist;
XtVaGetValues(outliner,
    XmNxrtGearNodeStyleList,    &stylelist,
    NULL);
```

The returned node style list is an XrtList object; the elements of this list are NodeStyle objects. (For details on XrtList objects, see section 7.2.3 on page 96.)

7.9.3 Icons

The NodeStyle object defines resources that specify the folder icons that appear on the screen when a tree is drawn. The following folder icon resources can be set:

`XmNxrtGearIconClosed`	The icon indicating a deselected closed folder.
`XmNxrtGearIconClosedSelected`	The icon indicating a closed folder that has been selected by the end-user.
`XmNxrtGearIconOpen`	The icon indicating a deselected open folder. (A folder is considered open if its state is anything but `XRTGEAR_FOLDERSTATE_CLOSED`.)
`XmNxrtGearIconOpenSelected`	The icon indicating a selected open folder.
`XmNxrtGearIconItem`	The icon indicating a deselected item.
`XmNxrtGearIconItemSelected`	The icon indicating a selected item.

Each of these resources accepts a pointer to a value of type `XrtGearIcon`, which is defined as follows:

```
typedef struct {
    Pixmap          icon;
    Pixmap          icon_mask;
} XrtGearIcon;
```

icon is the pixmap representing the icon to be displayed, and *icon_mask* is the optional clip mask that can be used to clip the icon. (For more information on clip masks, see section 2.4 on page 17, which describes the `XmNxrtGearMask` clip mask resource defined for Enhanced Label and Enhanced Pushbutton.)

The following code defines the deselected closed folder icon:

```
Widget          nodestyle;
Pixmap          closed_pixmap, closed_pixmap_mask;
XrtGearIcon     icon;

icon.icon = closed_pixmap;
icon.icon_mask = closed_pixmap_mask;
XtVaSetValues(nodestyle,
    XmNxrtGearIconClosed,   &icon,
    NULL);
```

The icons defined as part of the default node style are shown in Figure 62.



Figure 62 The default folder icons

Icon Spacing

Each NodeStyle object also defines an `XmNxrtGearLayoutSpacing` resource, which specifies the space in pixels between a node's icon and its label string.

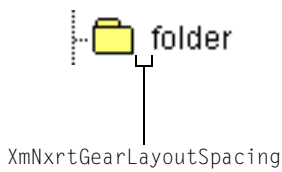


Figure 63 Icon spacing

The default value for `XmNxrtGearLayoutSpacing` is 5.

XPM Format

The *XrtGearIconLoadFromXpmData()* and *XrtGearIconLoadFromXpmFile()* convenience routines create `XrtGearIcon` structures from data objects in XPM format:

XrtGearIconLoadFromXpmData() loads an icon from an existing XPM data structure:

```
Widget              outliner;
XrtGearIcon         myicon;
String              my_xpm_data;
String              name;

XrtGearIconLoadFromXpmData(outliner, my_xpm_data, name,
    &myicon);
```

XrtGearIconLoadFromXpmFile() loads an icon from an existing XPM data file:

```
Widget              outliner;
XrtGearIcon         myicon;

XrtGearIconLoadFromXpmFile(outliner, "myicon.xpm", &myicon);
```

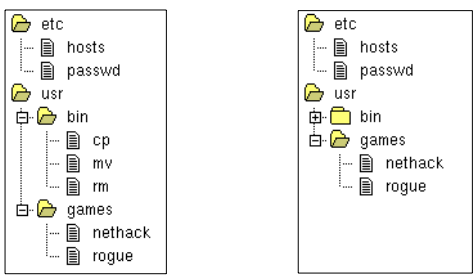
For more information on the XPM format, refer to the XPM Manual (not supplied).

7.9.4 Connecting Lines and Shortcut Buttons

The NodeStyle object defines resources which control the appearance and behavior of the lines drawn from folders to their items, and the appearance of the shortcut buttons.

A shortcut button is a small button associated with a particular folder. Each shortcut button contains a character indicating whether the folder is open (–) or closed (+). End-users can change the state of folders from open to closed, or vice versa, by clicking on the shortcut button. Figure 64 illustrates the behavior of shortcut buttons.

Note: changing line and shortcut node resources will only affect the children of the node, not the node itself.



folder open (shortcut button is –)

folder closed (shortcut button is +)

Figure 64 Shortcut button behavior

Connecting Lines

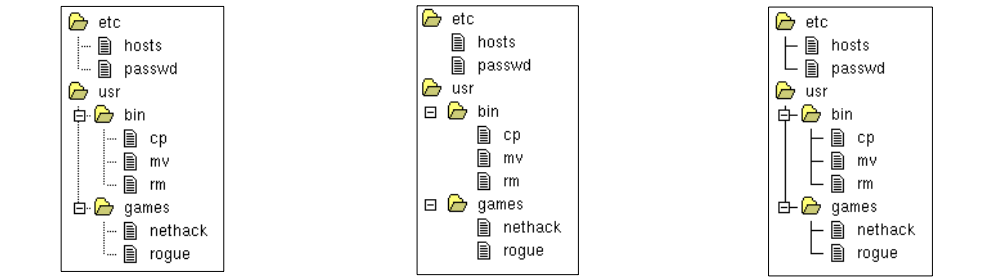
The following NodeStyle resources control the appearance of the connecting lines:

`XmNxrtGearLineColor` Specifies the color of the lines. By default, this color is inherited from `XmNforeground`.

`XmNxrtGearLineWidth` Specifies the width of the connecting lines. The default value for this resource is 1.

`XmNxrtGearLineStyle` Specifies the style of the connecting lines.

`XmNxrtGearLineStyle` can be set to any of the following values:



`XRTGEAR_LINESTYLE_DOTTED` (default)

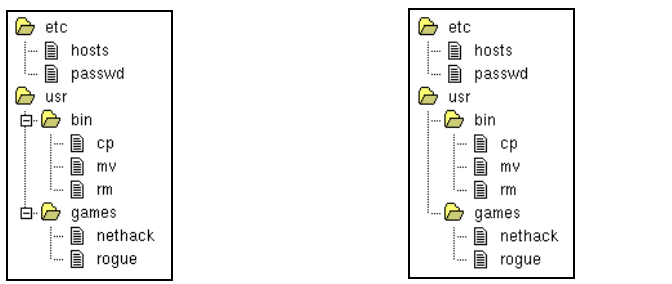
`XRTGEAR_LINESTYLE_NONE`

`XRTGEAR_LINESTYLE_SOLID`

Figure 65 Connecting line styles

Shortcut Buttons

The `XmNxrtGearShortcutShow` resource specifies whether shortcut buttons for the children of the folder are to be displayed. By default, `XmNxrtGearShortcutShow` is `True`, which means the buttons are displayed.

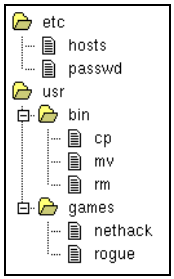


`XmNxrtGearShortcutShow = True`

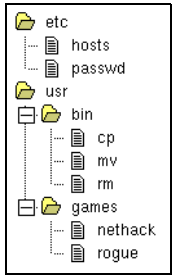
`XmNxrtGearShortcutShow = False`

Figure 66 Displaying shortcut buttons

The width and height of the shortcut buttons in pixels is specified by the `XmNxrtGearShortcutSize` resource. Its default value is 9 pixels. To achieve best results, set `XmNxrtGearShortcutSize` to an odd-numbered value if `XmNxrtGearLineWidth` is odd, and to an even-numbered value if `XmNxrtGearLineWidth` is even.



XmNxrtGearShortcutSize = 9



XmNxrtGearShortcutSize = 13

Figure 67 Shortcut button size

If `XmNxrtGearShortcutSize` is set to a value too large to display (greater than the value of either the `XmNxrtGearNodeHeight` or the `XmNrtGearNodeIndent` resource), the shortcut buttons are not displayed.

7.9.5 User-Defined Data

To associate data with a node, set the value of the node's `XmNuserData` resource:

```
Node        node;
XtPointer   mydata;

XtVaSetValues(node,
    XmNuserData,        mydata,
    NULL);
```

The Node object (or NodeFolder object) does not free any data pointed to by `XmNuserData`; you must do it yourself, just as with any Motif widget.

7.10 User Interaction

XRT/gear allows end-users to interact with Outliner widgets and their objects in the following ways:

- Selecting nodes and folders
- Resizing columns
- Opening and closing folder nodes
- Traversing the tree
- Dragging and dropping nodes
- Rearranging columns

Figure 68 illustrates the default user interaction, and lists the keys or mouse movements required. For a complete description of the translations and actions defined for the Outliner widget and its objects, see section 7.15 on page 198.

7.10.1 Selection Behavior

When a *Select(Start)* or *Select(Add)* action is performed, the current node is marked as selected. (By default, this occurs when the end-user presses the first mouse button.) This selection behavior in trees is controlled by two resources:

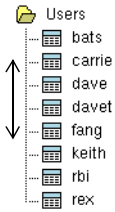
- `XmNxrtGearSelectionPolicy`, which specifies how many nodes can be selected at any one time
- `XmNxrtGearSelectionRestrictions`, which limits the types of nodes that can be selected.

`XmNxrtGearSelectionPolicy` specifies the selection mode currently in use. It can be set to any of the following values:

`XRTGEAR_SELECTION_POLICY_NONE`	Disallowed mode: no node selection permitted. Any node may be dragged.
`XRTGEAR_SELECTION_POLICY_SINGLE`	Single mode: only one node can be selected at a time; selection is toggled (re-selecting a selected node deselects it). Only the selected node may be dragged.

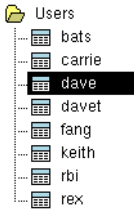
Moving Around in the Tree

- Press PgUp, PgDn, Home, End or arrow keys to move around in the tree
- Click on the scrollbars (if displayed) to move around in the tree



Selecting Nodes

Single mode:
Press MB1 to select the current node and unselect the others

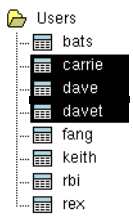


Disallowed mode:
No nodes are ever selected

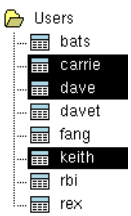
Single auto mode:
The current node is always selected

Multiple mode:
- Press MB1 or Ctrl+MB1 to start another selection range
- Press Shift+MB1 or drag MB1 to select all nodes between the bottom of the current range and the current node

Range mode:
- Press MB1 to select the current node and deselect the others
- Press Shift+MB1 or drag MB1 to select all nodes between the bottom of the current range and the current node

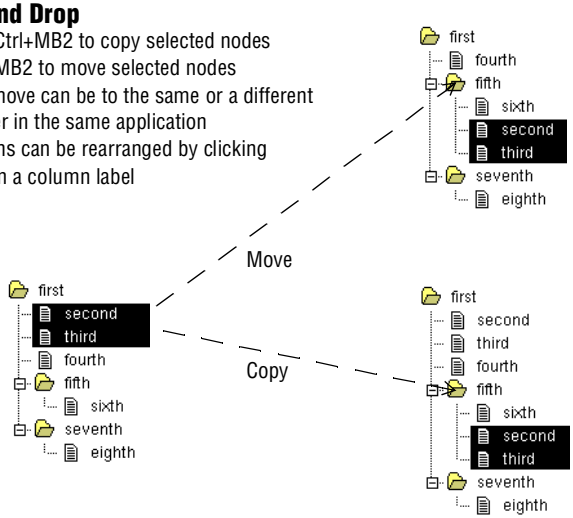


Multiple auto mode:
- Press MB1 to select the current node and deselect the others
- Press Shift+MB1 or drag MB1 to select all nodes between the bottom of the current range and the current node
- Press Ctrl+MB1 to start another selection range



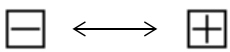
Drag and Drop

- Press Ctrl+MB2 to copy selected nodes
- Press MB2 to move selected nodes
- Copy/move can be to the same or a different Outliner in the same application
- Columns can be rearranged by clicking MB2 on a column label



Opening/Closing Folders

- Click MB1 on shortcut button
  or
- Double-click MB1 on folder



Column Resizing

Hold down MB1 on column border or column label border, drag the dashed line that appears

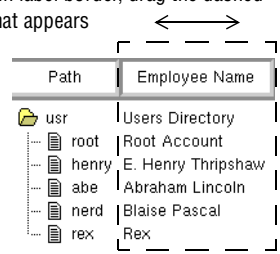


Figure 68 Default User Interactions

XRTGEAR_SELECTION_POLICY_SINGLE_AUTO_SELECT	Single Auto mode: only one node can be selected at a time; selection is not toggled (re-selecting a selected node does not deselect it). Only the selected node may be dragged.
XRTGEAR_SELECTION_POLICY_RANGE	Range mode: multiple nodes can be selected, provided they are in a contiguous range. Only the selected nodes may be dragged.
XRTGEAR_SELECTION_POLICY_MULTIPLE	Multiple mode: any number of nodes, contiguous or not, can be selected at any given time. Only the selected nodes may be dragged.
XRTGEAR_SELECTION_POLICY_MULTIPLE_AUTO_UNSELECT	Multiple Auto mode: clear all selected nodes before performing a selection/deselection operation on the current node. Only the selected nodes may be dragged.

The default value is XRTGEAR_SELECTION_POLICY_SINGLE_AUTO_SELECT.

Note that XmNxrtGearSelectionPolicy limits the node ranges that can be selected, but does not specify which nodes can be selected and which cannot. To provide restrictions on the types of nodes that can be selected, set the XmNxrtGearSelectionRestrictions resource. The value of this resource is one or more of the following constants ORed together:

XRTGEAR_RESTRICT_NOTHING	No restrictions; the end-user can select any node
XRTGEAR_RESTRICT_ITEMS	The end-user can only select folders, not items
XRTGEAR_RESTRICT_FOLDERS	The end-user can only select items, not folders
XRTGEAR_RESTRICT_MULTI_LEVEL	The end-user can only select siblings of previously selected nodes
XRTGEAR_RESTRICT_ALL	Selection is completely disabled

By default, XmNxrtGearSelectionRestrictions is set to XRTGEAR_RESTRICT_NOTHING.

If the value of XmNxrtGearSelectionPolicy or XmNxrtGearSelectionRestrictions is changed, any nodes selected by the end-user are automatically deselected.

Selecting a Node Programmatically

Normally, nodes get selected through user interaction (that is, users opening folders and clicking on nodes). If your application selects nodes programmatically, the node

may be contained in a closed folder. You need to insure that your application opens the folder so the node can be seen.

7.10.2 Difference Between Focus and Selected

A node that has the focus is not necessarily selected. The node that has the focus is stored in `XmNxrtGearNodeCurrent`. Outliner displays a border around the node with the focus.

A node that has been selected is stored in `XmNxrtGearSelectedNodeList`. Outliner displays a highlight bar around the selected node list.

7.10.3 Specifying Folder States

Normally, two states are defined for a folder: closed and open. A folder is closed if its children are not visible. A folder is open if its children are visible. By default, a folder state change (such as a double-click of the left mouse button) toggles between these two states.

You can also specify additional folder states that are somewhere between completely closed and completely open using the Outliner widget's `XmNxrtGearFolderStateMask` resource. The value of this resource is one or more of the following constants ORed together:

`XRTGEAR_FOLDERSTATE_CLOSED`	folder closed; no children visible
`XRTGEAR_FOLDERSTATE_OPEN_FOLDERS`	folder open; only folder children visible
`XRTGEAR_FOLDERSTATE_OPEN_ITEMS`	folder open; only item children visible
`XRTGEAR_FOLDERSTATE_OPEN_ALL`	folder open; all children visible

For example, the following example specifies the open, open folders only, and closed folder states:

```
XtVaSetValues(outliner,
    XmNxrtGearFolderStateMask, XRTGEAR_FOLDERSTATE_CLOSED |
    XRTGEAR_FOLDERSTATE_OPEN_FOLDERS | XRTGEAR_FOLDERSTATE_OPEN_ALL,
    NULL);
```

The default is `XRTGEAR_FOLDERSTATE_CLOSED | XRTGEAR_FOLDERSTATE_OPEN`.

If more than two states are defined, repeated folder state changes produce the following folder states in the order given (states not defined are skipped):

```
XRTGEAR_FOLDERSTATE_CLOSED
XRTGEAR_FOLDERSTATE_OPEN_FOLDERS or XRTGEAR_FOLDERSTATE_OPEN_ITEMS
XRTGEAR_FOLDERSTATE_OPEN_ALL
```

7.10.4 Retrieving the Selected Node List

To retrieve the list of the nodes currently selected by the end-user, get the value of the Outliner widget's XmNxrtGearSelectedNodeList resource:

```
Widget          outliner;
XrtGearObject   nodelist;

XtVaSetValues(outliner,
    XmNxrtGearSelectedNodeList, &nodelist,
    NULL);
```

The returned list is an XrtList object whose elements are Node (and/or NodeFolder) objects. XmNxrtGearSelectedNodeList is read-only.

To determine whether an individual node is selected, get the value of the node's XmNxrtGearSelected resource. If this resource is True, the node is selected. This resource is settable, which allows you to select nodes programmatically.

7.10.5 Selection Callbacks

You can specify callbacks to be invoked whenever a node is selected or deselected. To do this, add the callbacks to the list defined by the Outliner widget's XmNxrtGearSelectionCallback resource. Each callback is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;                 /* read-only */
    XEvent          *event;                 /* read-only */
    Widget          container;              /* read-only */
    Widget          node;
    Widget          selected_node_list;     /* read-only */
    Boolean         doit;
} XrtGearSelectCallbackStruct;
```

reason specifies the callback reason, and is one of the following:

XRTGEAR_REASON_SELECT_BEGIN	start selection operation
XRTGEAR_REASON_SELECT_END	end selection operation
XRTGEAR_REASON_UNSELECT_BEGIN	start deselection operation
XRTGEAR_REASON_UNSELECT_END	end deselection operation

When a node is selected, the selection callbacks are called as follows:

1. The callbacks are called on the newly selected nodes, with *reason* set to XRTGEAR_REASON_SELECT_BEGIN.
2. The callbacks are called on the nodes that become deselected as a result of this selection operation, with *reason* set to XRTGEAR_REASON_UNSELECT_BEGIN.
3. The callbacks are called again on the nodes that are becoming deselected, with *reason* set to XRTGEAR_REASON_UNSELECT_END.
4. The callbacks are called again on the newly selected nodes, with *reason* set to XRTGEAR_REASON_SELECT_END.

To suppress node selection, set *doit* to `False` when *reason* is `XRTGEAR_REASON_SELECT_BEGIN`.

If the `XmNxrtGearSelectionPolicy` resource is set to `XRTGEAR_SELECTION_POLICY_RANGE` or `XRTGEAR_SELECTION_POLICY_MULTIPLE`, you can also suppress node deselection. To do this, set *doit* to `False` when *reason* is `XRTGEAR_REASON_UNSELECT`.

7.10.6 Scrollbars

If a tree is too large to be displayed in the node area, scrollbars are displayed. The end-user can click on these scrollbars to specify the portion of the tree to be displayed in the node area. By default, horizontal scrollbars, vertical scrollbars, or both appear whenever appropriate. These are separate widgets which are defined as children of the Outliner widget.

The following Outliner resources control the behavior of these scrollbars:

- `XmNxrtGearScrollBarHorizDisplayPolicy` and `XmNxrtGearScrollBarVertDisplayPolicy` control when the horizontal and vertical scrollbars are to appear. Valid values for these resources are:

`XRTGEAR_DISPLAY_AS_NEEDED`	Show scrollbar only when needed (default)
`XRTGEAR_DISPLAY_ALWAYS`	Always show scrollbar
`XRTGEAR_DISPLAY_NEVER`	Never show scrollbar

- To retrieve the widget IDs of the horizontal and vertical scrollbars, get the values of the `XmNxrtGearScrollBarHoriz` and `XmNxrtGearScrollBarVert` resources. You can then set scrollbar resources (such as `XmNbackground`).

7.10.7 Drag and Drop

The Outliner widget provides the following drag and drop capabilities:

- The ability to move a subtree from an Outliner widget to any Outliner in the same application.
- The ability to import data from other widgets.

To enable subtree relocation, set the `XmNxrtGearNodeMovePolicy` resource. Valid values for this resource are the following:

`XRTGEAR_MOVE_NOWHERE`	Disallow movement of nodes (the default).
`XRTGEAR_MOVE_WITHIN_LEVEL`	Only allow nodes to be reordered with respect to their siblings.
`XRTGEAR_MOVE_WITHIN_APP`	Allow nodes to be moved to any other Outliner widget within the application.
`XRTGEAR_MOVE_WITHIN_WIDGET`	Allow nodes to be moved anywhere within the Outliner widget.

You can also drop to a location outside a tree; in this case, the node labels are converted to text.

To allow importing of data into a tree, set the `XmNxrtGearNodeImportPolicy` resource. Valid values are:

`XRTGEAR_IMPORT_NOTHING`	Do not allow importing of data into the tree (the default).
`XRTGEAR_IMPORT_ANYTHING`	Allow all importing of data, including foreign text.
`XRTGEAR_IMPORT_WITHIN_APP`	Allow importing of data from another Outliner in the same application.
`XRTGEAR_IMPORT_WITHIN_WIDGET`	Allow importing of data from the same widget (copies).

Imported data is turned into a subtree using the following criteria:

- Newlines separate nodes.
- Tab characters represent indent levels.
- Commas separate multiple labels for an individual node.

Figure 69 illustrates these criteria. *<nl>* represents a newline character, and *<tab>* represents a tab character. Note that the special characters used here correspond to the default node, indent and label separation characters used by *XrtGearNodeCreateTreeFromStream()*.

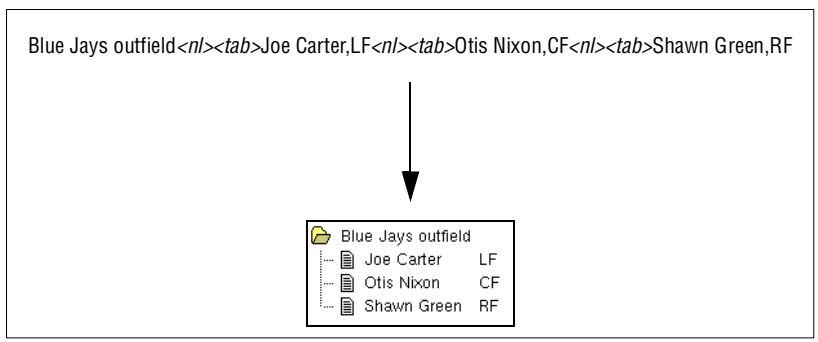


Figure 69 Converting data into a subtree

Once the subtree has been created from the imported data, it is inserted into the Outliner tree. The node currently pointed at by the end-user is treated as the target

node for the import operation. The exact imported behavior depends on the target node:

- If the target node is a folder, it becomes the parent of the new subtree.
- If the target node is an item and `XmNxrtGearAutoSort` is `False`, the target node's parent becomes the parent of the created subtree. The target node and the root of the subtree become siblings, with the root placed immediately before the target node.
- If the target node is an item and `XmNxrtGearAutoSort` is `True`, the target node's parent becomes the parent of the created subtree. The target node and the root of the subtree become siblings, with the siblings kept in sorted order.

Figure 70 shows the relationship between the imported subtree and the target node.

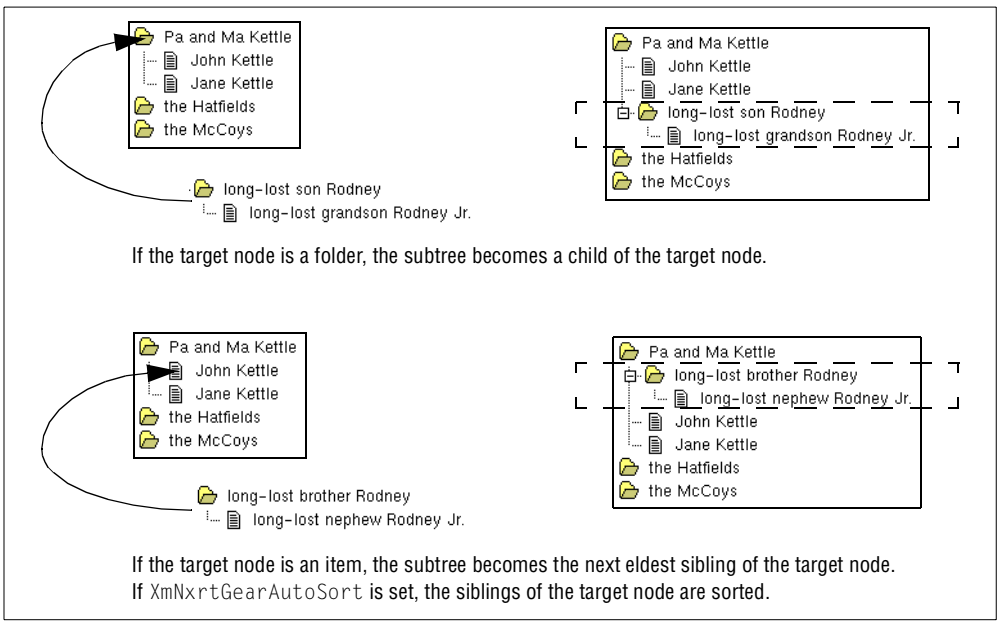


Figure 70 Imported data and the target node

Import Callbacks

You can supply callbacks to be invoked whenever data is imported into the tree or moved. To do this, add the callbacks to the list defined by the

XmNxrtGearImportCallback resource. Each callback in this list is passed the following structure:

```
typedef struct {
    XrtGearReason       reason;             /* read-only */
    XEvent              *event;             /* read-only */
    String              target;             /* read-only */
    String              data;               /* read-only */
    unsigned long       data_length;        /* read-only */
    Widget              object;             /* read-only */
    Boolean             doit;
} XrtGearOutlinerImportCallbackStruct;
```

The resulting subtree becomes a child of *object* if *object* is a folder. If *object* is a node, the subtree becomes a child of *object*'s parent.

Each callback in the XmNxrtGearImportCallback list is called twice: once before the import operation takes place (*reason* is XRTGEAR_REASON_IMPORT_BEGIN) and once after the import is complete (*reason* is XRTGEAR_REASON_IMPORT_END).

To suppress a particular import operation, set *doit* to False when *reason* is XRTGEAR_REASON_IMPORT_BEGIN.

7.10.8 Activate Callbacks

By default, the *Activate()* action is called whenever the end-user double-clicks on a folder or a column label, or presses Space or Return. You can specify activate callbacks to be invoked whenever the *Activate()* action is called. This enables you to write applications that update themselves in response to folder openings and closings.

To define an activate callback, add it to the callback list specified by the Outliner widget's XmNxrtGearActivateCallback resource. This resource is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          container;      /* read-only */
    Widget          node;
    int             click_count;
} XrtGearNodeActivateCallbackStruct;
```

container is the parent Outliner widget, and *node* is the node to be activated. *click_count* is the number of mouse clicks provided by the end-user.

7.10.9 Locating Events

If your application is detecting events (such as, for example, Expose events), the Outliner widget enables you to determine the node or column in which the event occurred.

To determine the node in which an event has occurred, call *XrtGearOutlinerNodeFromXEvent()*:

```
Widget              outliner, node;
XEvent              *event;
XrtGearLocation     location;

node = XrtGearOutlinerNodeFromXEvent(outliner, event, &location);
```

The return value for this method is the node over which (or near which) the event occurred; if this value is `NULL`, the event did not occur over a node. The third parameter, if non-`NULL`, specifies the returned position of the event, relative to the node. It can be any of the following:

`XRTGEAR_LOCATION_ON_NODE`	Event is over the node
`XRTGEAR_LOCATION_ON_SHORTCUT`	Event is on the node's shortcut button
`XRTGEAR_LOCATION_ON_LABEL`	Event is on a column label
`XRTGEAR_LOCATION_NOWHERE`	Event is not within any node

To determine the column in which an event has occurred, call *XrtGearColumnFromXEvent()*:

```
Widget              outliner, column;
XEvent              *event;

column = XrtGearColumnFromXEvent(outliner, event);
```

This method returns the XrtColumn object for the column over which the event occurred. It returns `NULL` if the event did not occur over a column.

7.10.10 Rearranging Columns

By default, the end-user can move a column from one location in an Outliner widget to another location in the same widget. To do this, the end-user presses the second mouse button while the cursor is positioned over a column, then releases the mouse button over the place where the column is to be relocated. (Note that the first column can never be moved.)

If you do not want to allow the end-user to rearrange columns, set the `XmNxrtGearColumnMoveAllow` resource to `False`. By default, this resource is `True`, which means column relocation is permitted.

Column Move Callbacks

You can specify callbacks to be called whenever a column is moved by the end-user. To do this, add the callbacks to the list defined by the

XmNxrtGearColumnMoveCallback resource. Each callback in this list is passed the following structure:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          column;             /* read-only */
    Widget          target_sibling;     /* read-only */
    Boolean         doit;
} XrtGearColumnMoveCallbackStruct;
```

column is the column to be moved; *target_sibling* is the column that will immediately precede the moved column after the move is complete. To suppress a column move, set *doit* to False when *reason* is XRTGEAR_REASON_COLUMN_MOVE_BEGIN.

Note that the ColumnMove callbacks are not called when *XrtGearOutlinerMoveColumns()* is called.

7.11 Printing Outliner Widgets

Many applications need to provide end-users with a way to get a printout of a tree they see on screen. XRT/gear can generate a PostScript image of an Outliner widget. This image is created with device-independent PostScript fonts and operators. The PostScript file can be printed on any PostScript printer or imported into other documents.

7.11.1 Generating PostScript Output

XRT/gear provides two methods that generate PostScript images of Outliner widgets: *XrtGearOutlinerDrawPS()* and *XrtGearOutlinerVaDrawPS()*. The two methods produce the same output, but provide different usage interfaces. *XrtGearOutlinerVaDrawPS()* uses a *VarArgs*-style interface (similar to *XtVaSetValues()*) to allow you to set only the arguments that you need to change from their defaults. *XrtGearOutlinerDrawPS()* uses an *ArgList*-style interface to enable you to maintain different sets of arguments for different printing requirements.

Defining Headers and Footers

To define headers and footers for the pages on which your widget is printed, set the XRTGEAR_PS_HEADER and XRTGEAR_PS_FOOTER attributes when calling *XrtGearOutlinerVaDrawPS()* or *XrtGearOutlinerDrawPS()*.

The fonts for these headers and footers are controlled by the XRTGEAR_PS_HEADER_FONT and XRTGEAR_PS_FOOTER_FONT attributes. The sizes of the fonts are controlled by the XRTGEAR_PS_HEADER_POINT_SIZE and XRTGEAR_PS_FOOTER_POINT_SIZE attributes; if not specified, the default is 12.0 points. The header and footer margins are controlled by XRTGEAR_PS_HEADER_MARGIN and XRTGEAR_PS_FOOTER_MARGIN.

Usage Tips

Following are some tips on using *XrtGearOutlinerVaDrawPS()* and *XrtGearOutlinerDrawPS()*:

- The XRTGEAR_PS_PAPERSIZE_ and XRTGEAR_PS_MARGIN_ attributes are always relative to the top of the *paper*, not the top of the *image*. That is, if the widget image orientation is landscape, the XRTGEAR_PS_PAPERSIZE_HEIGHT attribute still measures the height of the page, not the height of the image.
- To print multiple widgets on one page, explicitly position each widget on the page (so they don't overlap) with the XRTGEAR_PS_MARGIN_ attributes. Set XRTGEAR_PS_SHOWPAGE to False for every image except the last one.

XrtGearOutlinerVaDrawPS() Example

The following code outputs the first three levels of a tree to a file, using landscape mode.

```
FILE *fp;
fp = fopen("nodes.ps", "w");
if (fp) {
    numpages = XrtGearOutlinerVaDrawPS(outliner, fp,
        XRTGEAR_PS_ORIENTATION,     XRTGEAR_PS_LANDSCAPE,
        XRTGEAR_PS_START_NODE,      outliner,
        NULL);
    printf("%d page%s written.\n", numpages,
        (numpages==1)? "" : "s");
} else {
    printf("error opening file\n");
}
```

7.11.2 Customizing PostScript Fonts

Most X Servers provide enough information about the X fonts used in the widget to allow the PostScript output methods to automatically choose appropriate PostScript fonts. However, you can define a list of global PostScript to X font correspondences using the XmNxrtGearPSFontMapList resource. Each element of the list must be of type XrtGearPSFontMap, which is defined as follows:

```
typedef struct {
    String  font_name;
    String  tag;
    String  PS_font;
    double  point_size;
} XrtGearPSFontMap;
```

In this structure, *PS_font* is the PostScript font, and *point_size* is its point size. (Figure 71 lists the of standard PostScript font names.) The X font can be specified in either of two ways:

- You can use *font_name* to specify a complete X font name conforming to the X Logical Font Description Conventions (XLFD), such as "-*-times-bold-r-normal-*-140-*".
- You can use *tag* to specify an XmNfontList font tag (such as bold).

The list is terminated by defining an element with *PS_font* set to NULL.

If *point_size* is set to 0.0, the point size of the PostScript font is inferred from the point size of the corresponding X font.

Times-Roman	Helvetica	NewCenturySchlbk-Roman
Times-Bold	Helvetica-Bold	NewCenturySchlbk-Bold
Times-Italic	Helvetica-Oblique	NewCenturySchlbk-Italic
Times-BoldItalic	Helvetica-BoldOblique	NewCenturySchlbk-BoldItalic
Palatino-Roman	Bookman-Light	Helvetica-Narrow
Palatino-Bold	Bookman-Demi	Helvetica-Narrow-Bold
Palatino-Italic	Bookman-LightItalic	Helvetica-Narrow-Oblique
Palatino-BoldItalic	Bookman-DemiItalic	Helvetica-Narrow-BoldOblique
Courier	AvantGarde-Book	Symbol
Courier-Bold	AvantGarde-Demi	ZapfChancery-MediumItalic
Courier-Oblique	AvantGarde-BookOblique	ZapfDingbats
Courier-BoldOblique	AvantGarde-DemiOblique	

Figure 71 The 35 Standard LaserWriter II Font Names

The following example uses `XmNxrtGearPSFontMapList` to specify that Helvetica Narrow PostScript fonts are to correspond to Helvetica X fonts:

```
XrtGearPSFontMap fontlist[] = {
    {"-*-helvetica-medium-r-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow", 0.0},
    {"-*-helvetica-bold-r-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow-Bold", 0.0},
    {"-*-helvetica-medium-o-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow-Oblique", 0.0},
    {"-*-helvetica-bold-o-*-*-*-*-*-*-*-*-iso8859-1",
        NULL, "Helvetica-Narrow-BoldOblique", 0.0},
    {NULL, NULL, NULL, 0.0}
};
...
XtVaSetValues(outliner,
    XmNxrtGearPSFontMapList,    fontlist,
    NULL);
```

7.11.3 Print Callbacks

The `XmNxrtGearPrintCallback` resource specifies a list of callbacks that are called before and after each page of an outline is printed. These callbacks allow you to

monitor the printing process and change the output print file if desired. Each callback in this list is passed the following structure:

```
typedef struct {
    int         reason;             // Read-only
    XEvent      *event;             // Read-only
    int         current_page;       // Read-only
    int         total_pages;        // Read-only
    int         across_pages;       // Read-only
    int         down_pages;         // Read-only
    int         page_number;
    String      header, footer;
    FILE        *fp;
    Boolean     write_file_header;
    Boolean     write_file_trailer;
    Boolean     doit;               // initially True
} XrtGearPrintCallbackStruct;
```

reason is either `XRTGEAR_REASON_PAGE_PRINT_BEGIN` (before page printed) or `XRTGEAR_REASON_PAGE_PRINT_END` (after page printed). *current_page* is the page being printed, and *total_pages* is the total number of pages to print (measured as *across_pages* multiplied by *down_pages*).

The following members of this structure can be modified to allow you to control the outline printing process:

- *header* and *footer* are the page header and footer (specified by the call to *XrtGearOutlinerVaDrawPS()*).
- *page_number* contains the page number displayed in the header or footer.
- *fp* is the file to which the output is being written. (If you change this value to a new file, you will need to set *write_file_header* to `True` when you write to this new file for the first time. This tells XRT/gear to reprint the PostScript file header.) Changing *fp* enables you to, for example, write even and odd-numbered pages to separate files.
- If you are writing the last page of a file and the page is not the last page of the output, you will need to set *write_file_trailer* to `True`. This tells XRT/gear to write the PostScript file trailer. *write_file_trailer* is automatically set to `True` if *current_page* equals *total_page* (if the page being printed is the last page of the output).
- *doit* controls whether the current page is to be printed. To skip a page, set *doit* to `False` when *reason* is `XRTGEAR_REASON_PAGE_PRINT_BEGIN`. (Setting *doit* has no effect on *current_page* or *total_pages*.)

7.12 Performance Issues

7.12.1 Double Buffering

Double buffering is a graphics technique that reduces the amount of flashing perceived by an end-user when an Outliner widget changes or scrolls. Display updates are first drawn to an off-screen area, then copied to the screen.

By default, the Outliner widget implements double buffering. To turn it off, set the XmNxrtGearDoubleBuffer resource to False.

Note that double buffering requires more Server memory.

7.12.2 Controlling Repainting

By default, the Outliner widget is redrawn whenever it is changed. If you are changing several resources at once, you can improve performance by setting the XmNxrtGearRepaint resource to False before making the changes. This disables widget redrawing.

Once you have changed your resources, you can then re-enable widget repainting by setting XmNxrtGearRepaint back to its default value of True.

7.12.3 Fixed Column Widths

By default XmNxrtGearWidthSizingPolicy is set to XRTGEAR_SIZE_VARIABLE and calculates the column width based on the widest node label. You can increase performance by setting XmNxrtGearWidthSizingPolicy to either XRTGEAR_SIZE_PIXEL or XRTGEAR_SIZE_CHAR.

7.12.4 Use Lightweight Nodes

Regular folder and item nodes in Outliner are individual XtObjects. When creating very large Outliner applications, involving thousands of folder and item nodes, performance can suffer from having so many XtObjects. Outliner provides "lightweight" folder and item nodes. Lightweight nodes are non-XtObject nodes that can be added to an Outliner tree. See Lightweight Folder and Item Nodes on page 212 for more information.

7.12.5 Make Intelligent Use of the XmNxrtGearNodeCheckForDuplicates Resource

XmNxrtGearNodeCheckForDuplicates specifies whether or not Outliner should check the parent's child list for duplicate nodes. If True, Outliner scans for duplicate nodes. For operations where you are certain a node addition will not duplicate a node on the list, setting XmNxrtGearNodeCheckForDuplicates to False can increase Outliner's performance.

7.12.6 Do Not Create Trees with Complete Tree Hierarchy

XrtGearNodeCreateTreeFromBuffer() and XrtGearNodeCreateTreeFromStream() give you the option of creating trees where the complete tree hierarchy is provided for each node (*full_path* is True) or inferred from the indent levels (*full_path* is False). Creating a tree with *full_path* set to True can slow outliner. You should consider creating outliners with *full_path* set to False.

7.13 Resource Reference

This section lists all of the resources for the Outliner widget and its associated objects, plus some of the more commonly used inherited resources. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

7.13.1 Outliner Resources

XmNxrtGearActivateCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks to be called if the *Activate()* action is called. Each routine is passed a pointer to an `XrtGearNodeActivateCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to
                             an XrtGearNodeActivateCallbackStruct */
)
```

where `XrtGearNodeActivateCallbackStruct` is defined as

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          container;      /* read-only */
    Widget          node;
    int             click_count;
} XrtGearNodeActivateCallbackStruct;
```

reason is `XRTGEAR_REASON_ACTIVATE`. *event* is the XEvent associated with the *Activate()* action, or `NULL` if the callback is called programmatically; call *XrtGearNodeFromXEvent()* or *XrtGearColumnFromXEvent()* to retrieve the node or column in which the XEvent occurred. *container* is the parent Outliner widget. *node* is the node to be activated. *click_count* is the number of mouse clicks performed by the end-user.

Note that this resource is a user information callback only and does not indicate any special action on the part of the Outliner widget.

XmNxrtGearAutoSort **Boolean** **False** **CSG**

Specifies whether to automatically sort nodes against their siblings. Nodes are automatically resorted when this resource is changed to `True`.

XmNbackground **Pixel** ***inherited*** **CSG**

Specifies the widget background color.

XmNxrtGearBorderLocation **XrtGearBorderLocation** **_EACH_LABEL_SEPARATED_FROM_NODES** **CSG**

Specifies where the border is drawn. Valid values are:

`XRTGEAR_BORDER_LOC_LABELS_SEPARATED_FROM_NODES`
: Draw separate borders around the column label area and the node space.

XRTGEAR_BORDER_LOC_EACH_LABEL_SEPARATED_FROM_NODES
: Draw separate borders around the column label area and the node space, and draw a border around each individual column label.

XRTGEAR_BORDER_LOC_LABELS
: Only draw a border around the column label area.

XRTGEAR_BORDER_LOC_EACH_LABEL
: Draw a border around each column label individually.

XRTGEAR_BORDER_LOC_NODES
: Only draw a border around the node space.

XRTGEAR_BORDER_LOC_LABELS_AND_NODES
: Draw a contiguous border around the combined areas of the column label area and the node space.

XmNxrtGearBorderType **XrtGearBorderType** **XRTGEAR_BORDER_IN** **CSG**

Specifies the style used for the Outliner border. Valid values are:

XRTGEAR_BORDER_PLAIN	border drawn in foreground color
XRTGEAR_BORDER_IN	label appears inset
XRTGEAR_BORDER_OUT	label appears raised
XRTGEAR_BORDER_ETCHED_IN	double line; label appears inset
XRTGEAR_BORDER_ETCHED_OUT	double line; label appears raised
XRTGEAR_BORDER_BEVEL	1-pixel in border drawn at 1/2 shadow thickness
XRTGEAR_BORDER_FRAME_IN	1-pixel shadow-in at edge; label appears framed
XRTGEAR_BORDER_FRAME_OUT	1-pixel shadow-out at edge; label appears framed
XRTGEAR_BORDER_NONE	no border

The border width is specified by XmNxrtGearShadowThickness.

XmNxrtGearColumnLabelsShow **Boolean** **False** **CSG**

Specifies whether column labels are to be displayed.

XmNxrtGearColumnList **XrtGearObject** **NULL** **G**

Specifies a list of all column objects defined for the Outliner widget.

XmNxrtGearColumnMoveAllow **Boolean** **True** **CSG**

Specifies whether column movement is permitted. The first column can never be moved.

XmNxrtGearColumnMoveCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks to be called if a column is moved by the end-user. Each routine is passed a pointer to an `XrtGearColumnMoveCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to
                             an XrtGearColumnMoveCallbackStruct */
)
```

where `XrtGearColumnMoveCallbackStruct` is defined as

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          column;             /* read-only */
    Widget          target_sibling;     /* read-only */
    Boolean         doit;
} XrtGearColumnMoveCallbackStruct;
```

reason is `XRTGEAR_REASON_COLUMN_MOVE_BEGIN` (before column move) or `XRTGEAR_REASON_COLUMN_MOVE_END` (after column move). *event* is `NULL`. *column* is the column to be moved. *target_sibling* is the column that will immediately precede *column* after the move operation is complete. *doit* indicates whether the column move operation is to be performed; to suppress the column move, set *doit* to `False` when *reason* is `XRTGEAR_REASON_COLUMN_MOVE_BEGIN`.

The ColumnMove callbacks are not called when the application calls *XrtGearOutlinerMoveColumns()*.

XmNxrtGearColumnOrderList **XrtGearObject** **NULL** **G**

Specifies the list of columns in the order in which they are displayed. To rearrange the display order, call *XrtGearOutlinerMoveColumns()*.

XmNxrtGearColumnSelected **Widget** **NULL** **CSG**

Specifies a pointer to the currently-selected column.

XmNxrtGearColumnSelectionCallback **Widget** **XtCallbackList** **CSG**

Specifies the callbacks to call when a column selection occurs. Each routine is passed a pointer to an `XrtGearColumnSelectionCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to
                             an XrtGearColumnSelectionCallbackStruct */
)
```

where XrtGearColumnSelectionCallbackStruct is defined as

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          old_column;         /* read-only */
    Widget          new_column;
    Boolean         doit;
} XrtGearColumnSelectionCallbackStruct;
```

reason can be XRTGEAR_REASON_SELECT_BEGIN (selection about to begin), XRTGEAR_REASON_SELECT_END (selection complete), XRTGEAR_REASON_UNSELECT_BEGIN (deselection about to begin), or XRTGEAR_REASON_UNSELECT_END (deselection about to end). *event* is the XEvent that triggered the callback, or NULL if the callback is called programmatically. *old_column* is the previous column; it is read-only, and may be NULL. *new_column* is the selected column, and can be modified if *reason* is XRTGEAR_REASON_SELECT_BEGIN. *doit* specifies whether the column selection is to be permitted; this can be modified if *reason* is XRTGEAR_REASON_SELECT_BEGIN.

The callbacks specified by XmNxrtGearColumnSelectionCallback are called in the following order, with the following values as the *reason* member of the XrtGearColumnSelectionCallbackStruct structure:

XRTGEAR_REASON_SELECT_BEGIN	on the newly selected node
XRTGEAR_REASON_UNSELECT_BEGIN	on the previously selected node
XRTGEAR_REASON_UNSELECT_END	on the previously selected node
XRTGEAR_REASON_SELECT_END	on the newly selected node

XmNxrtGearCursor **Cursor** **XC_left_ptr** **CSG**

Specifies the mouse cursor used for normal point-and-click operations of the Outliner.

XmNxrtGearCursorResizeWidth **Cursor** **XC_right_side** **CSG**

Specifies the mouse cursor used when resizing the width of a column.

XmNxrtGearCursorTracking **Boolean** **True** **CSG**

Specifies whether to change the mouse cursor as it moves over the different sensitive areas and call the callback routines specified by XmNxrtGearCursorTrackingCallback. If True, the cursor is changed and the callbacks are called.

XmNxrtGearCursorTrackingCallback **XtCallbackList** **NULL** **CSG**

Specifies the callbacks to be invoked if XmNxrtGearCursorTracking is set to True and the cursor moves over a sensitive area. Each routine is passed a pointer to an

XrtGearCursorTrackingCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearCursorTrackingCallbackStruct */
)
```

where XrtGearCursorTrackingCallbackStruct is defined as

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          object;         /* read-only */
    Cursor          old_cursor;     /* read-only */
    Cursor          new_cursor;
} XrtGearCursorTrackingCallbackStruct;
```

reason is one of the following:

XRTGEAR_REASON_CURSOR_EDGE_LEFT	cursor moves over left edge
XRTGEAR_REASON_CURSOR_EDGE_RIGHT	cursor moves over right edge
XRTGEAR_REASON_CURSOR_INSIDE	cursor moves inside node or label
XRTGEAR_REASON_CURSOR_LEAVE	cursor leaves node or label

event is the XEvent that triggered the callback, or NULL if the callback is called programmatically. *object* is the node or column label the cursor is currently on. *old_cursor* is the appearance of the cursor before it reached the sensitive area that triggered the callback. *new_cursor* is the new appearance of the cursor, and can be changed.

XmNxrtGearDoubleBuffer **Boolean** **True** **CSG**

Specifies whether or not the widget should use double-buffering for better performance. This will require the use of more Server memory.

XmNeditable **Boolean** **True** **CSG**

This resource controls whether or not users are allowed to edit node labels in the outliner. If set to False, this overrides the editability setting of nodes, folders and columns.

XmNxrtGearEditColumn **int** **XRTGEAR_NOVALUE** **G**

Specifies the index of the column in which the node is being edited, or XRTGEAR_NOVALUE if no edit is currently in progress.

XmNxrtGearEditNode **Widget** **NULL** **G**

Specifies the widget ID of the node currently being edited, or NULL if no node is currently being edited.

XmNxrtGearEditor **Widget** **NULL** **G**

Specifies the widget ID of the current widget used for cell editing. The resource is read-only and is NULL if no cell is currently being edited. In this release, only an XmText widget can be used for editing a cell.

XmNxrtGearFolderStateCallback **XtCallbackList** **NULL** **CSG**

Called when a folder node changes state via the *ChangeState()* action or the *XrtGearNodeChangeFolderState()* method. Each callback is passed a pointer to an XrtGearNodeFolderStateCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearNodeFolderStateCallbackStruct */
)
```

where XrtGearNodeFolderStateCallbackStruct is defined as

```
typedef struct {
    XrtGearReason       reason;     /* read-only */
    XEvent              *event;     /* read-only */
    Widget              container;  /* read-only */
    Widget              node;
    XrtGearFolderState  new_state;
    XrtGearFolderState  old_state;  /* read-only */
    Boolean             doit;
} XrtGearNodeFolderStateCallbackStruct;
```

reason is either XRTGEAR_REASON_FOLDER_CHANGE_BEGIN or XRTGEAR_REASON_FOLDER_CHANGE_END. *event* is the XEvent that triggered the callback, or NULL if the callback is called programmatically. *container* is the parent Outliner widget. *node* is the folder node whose state is to change. *new_state* is the new state of the folder. *old_state* is the old state of the folder. Setting *doit* to False (if the *reason* is XRTGEAR_REASON_FOLDER_CHANGE_BEGIN) prevents the folder changing states.

XmNxrtGearFolderStateMask **int** **XRTGEAR_FOLDERSTATE_OPEN_ALL | _CLOSED** **CSG**

Controls the permitted combination of states for folders. Valid values are OR'ed combinations of the following:

XRTGEAR_FOLDERSTATE_CLOSED	folder closed; no children visible
XRTGEAR_FOLDERSTATE_OPEN_FOLDERS	folder open; only folder children visible
XRTGEAR_FOLDERSTATE_OPEN_ITEMS	folder open; only item children visible
XRTGEAR_FOLDERSTATE_OPEN_ALL	folder open; all children visible

At least one state must be specified. XRTGEAR_FOLDERSTATE_OPEN_FOLDERS and XRTGEAR_FOLDERSTATE_OPEN_ITEMS are mutually exclusive; you can only specify one or the other. Clicking on a folder node changes state in the following cycle:

```
XRTGEAR_FOLDERSTATE_CLOSED
XRTGEAR_FOLDERSTATE_OPEN_FOLDERS, XRTGEAR_FOLDERSTATE_OPEN_ITEMS
XRTGEAR_FOLDERSTATE_OPEN_ALL
```

XmNfontList **XmFontList** ***inherited*** **CSG**

Specifies the default font for all nodes. Also used to calculate the width used by XmNxrtGearHeightPreferredCount and XmNxrtGearWidthChar.

XmNforeground **Pixel** ***inherited*** **CSG**

Specifies the default foreground color. The shortcut buttons are always drawn in this color.

XmNheight **Dimension** ***determined at run-time*** **CSG**

Specifies the height of the widget in pixels.

XmNxrtGearHeightPreferredCount **int** **10** **CSG**

This resource is used to calculate the preferred height of the displayed portion of the tree when XmNxrtGearHeightSizingPolicy is set to XRTGEAR_SIZE_NODE or XRTGEAR_SIZE_CHAR.

If XmNxrtGearHeightSizingPolicy is set to XRTGEAR_SIZE_NODE, the value of XmNxrtGearHeightPreferredCount is multiplied by the value of XmNxrtGearNodeHeight to yield the preferred height of the displayed portion of the tree.

If XmNxrtGearHeightSizingPolicy is set to XRTGEAR_SIZE_CHAR, the value of XmNxrtGearHeightPreferredCount is multiplied by the height of the highest font in the XmNfontList to yield the preferred height of the displayed portion of the tree.

XmNxrtGearHeightResizePolicy **XrtGearWidgetResizePolicy** **XRTGEAR_WIDGET_RESIZE_VARIABLE** **CSG**

Specifies when a widget should try to resize itself. Valid values are:

XRTGEAR_WIDGET_RESIZE_VARIABLE	Resize to the preferred height determined by XmNxrtGearHeightSizingPolicy.
XRTGEAR_WIDGET_RESIZE_GROW_ONLY	Resize to the preferred height if the widget grows as a consequence; otherwise, leave unchanged.
XRTGEAR_WIDGET_RESIZE_FIXED	Don't resize unless a parent requests it.

XmNxrtGearHeightSizingPolicy **XrtGearSizingPolicy** **XRTGEAR_SIZE_NODE** **CSG**

Specifies the manner in which the preferred height of the displayed portion of the tree is calculated. Valid values are:

XRTGEAR_SIZE_PIXEL	Use the value of XmNxrtGearHeightPreferredCount to specify the preferred height in pixels.
XRTGEAR_SIZE_CHAR	Use the height of the highest font in the list specified by XmNfontList multiplied by the value of XmNxrtGearHeightPreferredCount to calculate the preferred height.
XRTGEAR_SIZE_VARIABLE	Use the number of non-collapsed nodes to calculate the preferred height.
XRTGEAR_SIZE_NODE	Use the following formula to calculate the preferred height:

```
XmNxrtGearHeightPreferredCount * XmNxrtGearNodeHeight +
(XmNxrtGearHeightPreferredCount - 1) * XmNxrtGearNodeSpacing +
XmNhighlightThickness * 2
```

XmNxrtGearHeightVirtual **int** ***dynamic*** **G**

Specifies the virtual height of the widget in pixels. This is maintained for use by the vertical scrollbar. If the height is greater than ULONG_MAX, this value will be set to ULONG_MAX.

XmNhighlightColor **Pixel** ***inherited*** **CSG**

Specifies the color of the outer half of the focus highlight drawn around the current node (specified by XmNxrtGearNodeCurrent). (The inner half of the focus highlight is drawn in the background color, specified by XmNbackground.)

XmNhighlightThickness **Dimension** **2** **CSG**

Specifies the thickness of the focus highlight drawn around the current node. This focus highlight is drawn in two colors: the outer half of the focus highlight is drawn in the focus highlight color (specified by XmNhighlightColor), and the inner half is drawn in the background color (specified by XmNbackground).

If XmNhighlightThickness is set to an odd-numbered value, the outer half of the focus highlight is one pixel larger than the inner half.

This resource must be set to a value greater than zero.

XmNxrtGearImportCallback **XtCallbackList** **NULL** **CSG**

Specifies callbacks that are called whenever data is imported or moved into an Outliner widget. Each routine is passed a pointer to an XrtGearOutlinerImportCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearOutlinerImportCallbackStruct */
)
```

where XrtGearOutlinerImportCallbackStruct is defined as

```
typedef struct {
    XrtGearReason       reason;             /* read-only */
    XEvent              *event;             /* read-only */
    String              target;             /* read-only */
    String              data;               /* read-only */
    unsigned long       data_length;        /* read-only */
    Widget              object;             /* read-only */
    Boolean             doit;
} XrtGearOutlinerImportCallbackStruct;
```

reason is either XRTGEAR_REASON_IMPORT_BEGIN (before import operation) or XRTGEAR_REASON_IMPORT_END (after import operation). *event* is NULL. *target* is the name of the type of the data being imported. *data* is the data being imported; newlines separate nodes of the subtree created from the imported data, tabs specify indent levels, and multiple labels are separated by commas. *data_length* is the length of the imported data.

object is the target node or column specified by the end-user. If *object* is a folder, it becomes the parent of the imported subtree. If *object* is a node, the root of the subtree becomes a sibling of

object. (If XmNxrtGearAutoSort is True, the root of the sibling is placed in the proper sorted order.)

The import operation can be suppressed by setting *doit* to False when *reason* is XRTGEAR_REASON_IMPORT_BEGIN.

XmNxrtGearMarginHeight **Dimension** **5** **CSG**

Specifies the space between the top and bottom edges of the widget and its border.

XmNxrtGearMarginWidth **Dimension** **5** **CSG**

Specifies the space between the left and right edges of the widget and its border.

XmNxrtGearNodeCheckForDuplicates **Boolean** **True** **CSG**

Specifies whether or not Outliner should check the parent's child list for duplicate nodes. If True, Outliner scans for duplicate nodes. For operations where you are certain a node addition will not duplicate a node on the list, setting XmNxrtGearNodeCheckForDuplicates to False can increase Outliner's performance.

XmNxrtGearNodeChildList **XrtGearObject** **NULL** **G**

Specifies a pointer to an XrtList object that contains the nodes to be drawn (the children of the Outliner widget, which serves as the root node of the tree).

XmNxrtGearNodeCompareProc **XrtGearListItemCompareProc** **NULL** **CSG**

Specifies the function the widget calls to compare two nodes for sorting purposes. Default functions perform a standard alphanumeric compare of the labels of the nodes. Additionally, if the XmNxrtGearColumnSortOrderList resource is set, they can perform a multi-key sort.

The sorting procedure is of the following form:

```
int compare(XtPointer node1, XtPointer node2)
```

The function returns negative if node1 < node2, 0 if node1 = node2, and positive if node1 > node2.

XmNxrtGearNodeCreateCallback **XtCallbackList** **NULL** **CSG**

Specifies the callbacks called when the Outliner widget creates a node in the list. This will most likely happen as a result of a *Drag(Copy)* action. Each routine is passed a pointer to an XrtGearNodeCreateCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearNodeCreateCallbackStruct */
)
```

where XrtGearNodeCreateCallbackStruct is defined as

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          source_node;    /* read-only */
    XrtGearObject   label;
    Widget          node_style;
    XtPointer       user_data;
    Widget          new_node;
} XrtGearNodeCreateCallbackStruct;
```

reason is either XRTGEAR_REASON_NODE_CREATE_BEGIN (before node created) or XRTGEAR_REASON_NODE_CREATE_END (after node created). *event* is NULL. If the new node is being created as the result of a copy, *source_node* is the node being copied.

If *reason* is XRTGEAR_REASON_NODE_CREATE_BEGIN, *label*, *node_style*, *user_data* and *new_node* are NULL, but are writeable. Setting *label*, *node_style* or user_data enables you to define the label, node style, or user-defined data for your new node; setting new_node enables you to define the new node yourself. If new_node is set, the callbacks are not called with reason XRTGEAR_REASON_NODE_CREATE_END.

If *reason* is XRTGEAR_REASON_NODE_CREATE_END, *new_node* points to the newly-created node. *label*, *node_style* and *user_data* point to the newly created label, node style and user-defined data; these values cannot be modified at this point.

Note that these callbacks are not called when a node is created programmatically (e.g. by calling *XrtGearNodeCreateTreeFromStream()*).

XmNxrtGearNodeCurrent **Widget** **NULL** **CSG**

Specifies a pointer to the node that currently has focus. If this is set by the end-user, it will cause the new current node to become visible.

Changing this resource does not trigger the callbacks specified by XmNxrtGearNodeCurrentChangedCallback.

XmNxrtGearNodeCurrentChangedCallback **XtCallbackList** **NULL** **CSG**

Specifies the callbacks called when the Outliner widget's current node changes as the result of a user action or a call to *XrtGearNodeTraverseTo()*. (Changing XmNxrtGearNodeCurrent does not trigger these callbacks.) Each routine is passed a pointer to an XrtGearNodeCurrentChangedCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearNodeCurrentChangedCallbackStruct */
)
```

where XrtGearNodeCurrentChangedCallbackStruct is defined as

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    Widget          container;      /* read-only */
    Widget          previous_node;  /* read-only */
    Widget          new_node;
    Boolean         doit;
} XrtGearNodeCurrentChangedCallbackStruct;
```

event is the XEvent that triggered the callback, or NULL if the callback is called programmatically. *reason* is either XRTGEAR_REASON_NODE_CHANGE_BEGIN or XRTGEAR_REASON_NODE_CHANGE_END. *container* is the parent Outliner widget. *previous_node* is the previous current node, and *new_node* is the new current node. If *reason* is XRTGEAR_REASON_NODE_CHANGE_BEGIN, you may set *new_node* to refer to any widget you like, or set *doit* to False to abort the node change operation. (By default, *doit* is True.)

If *reason* is XRTGEAR_REASON_NODE_CHANGE_END, *new_node* points to the new current node and cannot be modified, and *doit* has no meaning.

XmNxrtGearNodeEnterCellCallback XtCallbackList NULL CSG

Specifies the list of callbacks which are called before and after completing a traversal from one cell to another. A pointer to an XrtGearNodeEnterCellCallbackStruct is passed to each callback. The form of the callback is:

```
void
XrtGearNodeEnterCellCallback (
   Widget      outliner,
   XtPointer   client_data
   XtPointer   call_data /* points to an
                            XrtGearNodeEnterCellCallbackStruct */
)
```

where XrtGearNodeEnterCellCallbackStruct is defined as:

```
typedef struct {
   XrtGearReason   reason; /* read only */
   XEvent          *event; /* read only */
   Widget          previous_node; /* read only */
   int             previous_column; /* read only */
   Widget          editor; /* Widget used as the editor */
   Widget          new_node;
   int             new_column;
   Boolean         doit; /* initally True */
} XrtGearNodeEnterCellCallbackStruct;
```

reason is either XRTGEAR_REASON_NODE_ENTER_CELL_BEGIN (before the cell is entered) or XRTGEAR_REASON_NODE_ENTER_CELL_END (after the text widget has been mapped to the new cell, notification only) ; *previous_node* and *previous_column* indicate where the text widget is coming from (NULL and XRTGEAR_NOVALUE respectively if no cell is currently being edited); *new_node* and *new_column* indicate the cell being traversed to; *doit* is a flag that determines whether to allow the cell to be entered. To prevent the user from entering this cell, set *doit* to False when *reason* is XRTGEAR_REASON_ENTER_CELL_BEGIN. To change the destination cell, change *new_node* and *new_column* when *reason* is XRTGEAR_REASON_ENTER_CELL_BEGIN.

XmNxrtGearNodeHeight **Dimension** **20** **CSG**

Specifies the height of a node. By default, this is the maximum of the heights of the node styles.

XmNxrtGearNodeHighlightOnDrag **Boolean** **False** **CSG**

When set to `True`, the node that the mouse pointer is currently over is highlighted (with a focus rectangle) during drag-and-drop operations. This can help the user identify which node will be the target when the mouse button is released.

XmNxrtGearNodeImportPolicy **XrtGearImportPolicy** **XRTGEAR_IMPORT_NOTHING** **CSG**

Specifies the restrictions on importing data into the Outliner widget. Valid values are:

`XRTGEAR_IMPORT_NOTHING`	Do not allow importing of data into the tree (the default).
`XRTGEAR_IMPORT_ANYTHING`	Allow all importing of data, including foreign text.
`XRTGEAR_IMPORT_WITHIN_APP`	Allow importing of data from another Outliner in the same application.
`XRTGEAR_IMPORT_WITHIN_WIDGET`	Allow importing of data from the same widget (copies).

XmNxrtGearNodeIndent **int** **20** **CSG**

Specifies the amount of indenting in pixels for each hierarchy level. It cannot be a negative value.

XmNxrtGearNodeMoveCallback **XtCallbackList** **NULL** **CSG**

Specifies callbacks that are called any time a node is moved or reparented. These callbacks are invoked each time an end-user moves a node interactively. Each routine is passed a pointer to an `XrtGearNodeMoveCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearNodeMoveCallbackStruct */
)
```

where `XrtGearNodeMoveCallbackStruct` is defined as

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    XEvent          *event;         /* read-only */
    unsigned char   operation;      /* read-only */
    Widget          node;           /* read-only */
    Widget          old_parent;     /* read-only */
    Widget          new_parent;     /* read-only */
    Widget          new_next;
    Boolean         doit;
} XrtGearNodeMoveCallbackStruct;
```

reason is XRTGEAR_REASON_NODE_MOVE_BEGIN, XRTGEAR_REASON_NODE_MOVE_END XRTGEAR_REASON_NODE_DRAG_BEGIN or XRTGEAR_REASON_NODE_DRAG_END. *event* is NULL. *operation* is either XmDROP_MOVE (operation is a move) or XmDROP_COPY (operation is a copy). *node* is the node being moved. *old_parent* is the existing parent of the node (if the node is being reparented). *new_parent* is the new parent of the node after reparenting. *new_next* is a child of *new_parent*, and, if not NULL, represents the next-youngest sibling of *node* after the move; if XmNxrtGearAutoSort is False, you can modify *new_next* to change the placement of *node* in the list of the children of *new_parent*.

The move can be aborted when *reason* is XRTGEAR_REASON_NODE_MOVE_BEGIN. To do this, set *doit* to False. The callbacks are called with *reason* set to XRTGEAR_REASON_NODE_MOVE_END after the move but before the redraw takes place.

XmNxrtGearNodeMovePolicy **XrtGearMovePolicy** **XRTGEAR_MOVE_NOWHERE** **CSG**

Specifies whether to allow end-users to drag and drop within the Outliner widget. Valid values are:

XRTGEAR_MOVE_NOWHERE	Disallows movement of nodes.
XRTGEAR_MOVE_WITHIN_LEVEL	Only allows nodes to be reordered with respect to their siblings.
XRTGEAR_MOVE_WITHIN_WIDGET	Allows nodes to be moved anywhere within the Outliner widget.
XRTGEAR_MOVE_WITHIN_APP	Allows nodes to be moved to any other Outliner within the application.

XmNxrtGearNodeSpacing **Dimension** **2** **CSG**

Specifies the space between the top and bottom of any two consecutive nodes. This must be greater than or equal to the value of XmNhighlightThickness (which in turn must be at least 1). This restriction ensures that there is always enough room to draw the focus highlight.

XmNxrtGearNodeStyleDefault **Widget** ***default node style*** **CSG**

Specifies the NodeStyle object to be used as the default node style. If this resource is not set, the first element in the list specified by XmNxrtGearNodeStyleList is used as the default node style.

XmNxrtGearNodeStyleList **XrtGearObject** ***contains default node style*** **G**

Specifies a pointer to an XrtList of NodeStyle objects. If XmNxrtGearNodeStyleDefault is not set, the first element in this list is assumed to be the default node style, and is used by all nodes whose XmNxrtGearNodeStyle resource is not set.

XmNxrtGearNodeTop **Widget** ***dynamic*** **CSG**

Specifies a pointer to the topmost visible node. This value can also be obtained from the list specified by XmNxrtGearNodeVisibleList.

XmNxrtGearNodeValidateCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks which are called when a node label has been edited, before and after committing the edit. This callback is triggered by the *CommitEdit()* action. A pointer to an `XrtGearValidateNodeCallbackStruct` is passed to each callback. The form of this callback is:

```
void
ValidateCallback (
    Widget      outliner,
    XtPointer   client_data
    XtPointer   call_data // points to an
                              XrtGearNodeValidateCallbackStruct
)
```

where `XrtGearNodeValidateCallbackStruct` is defined as:

```
typedef struct {
    XrtGearReason   reason; /* read only */
    XEvent          *event; /* read only, currently NULL */
    Widget          node; /* read only */
    XrtGearObject   old_label; /* read only */
    int             column; /* read only */
    String          value;
    Boolean         doit; /* initially True */
    Boolean         bell; /* initially False */
} XrtGearNodeValidateCallbackStruct;
```

reason is `XRTGEAR_REASON_NODE_VALIDATE_BEGIN` (before commiting the edit) or `XRTGEAR_REASON_NODE_VALIDATE_END` (after commiting the edit). *event* is `Null` if this is the result of a call to *XrtGearOutlinerCommitEdit(). old_label* is the label of this node before any changes were made. *column* is the column the edited node label is in. *value* contains the data to be commited if *doit* is `True`. *doit* should be set to `False` if the value should not be commited (this only applies when *reason* is `XRTGEAR_REASON_NODE_VALIDATE_BEGIN`). *bell* indicates whether to ring the terminal bell to indiate that the edited cell has not been commited.

XmNxrtGearNodeVisibleList **XrtGearObject** ***dynamic*** **G**

Specifies a pointer to an XrtList object, which contains a list of the currently visible nodes.

XmNxrtGearPrintCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called as each page of a table is printed. Each routine is passed a pointer to an `XrtGearPrintCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearPrintCallbackStruct */
)
```

where XrtGearPrintCallbackStruct is defined as

```
typedef struct {
    int             reason;             /* read-only */
    XEvent          *event;             /* read-only */
    int             current_page;       /* read-only */
    int             total_pages;        /* read-only */
    int             across_pages;       /* read-only */
    int             down_pages;         /* read-only */
    int             page_number;
    String          header, footer;
    FILE            *fp;
    Boolean         write_file_header;
    Boolean         write_file_trailer;
    Boolean         doit;
} XrtGearPrintCallbackStruct;
```

reason is either XRTGEAR_REASON_PAGE_PRINT_BEGIN or XRTGEAR_REASON_PAGE_PRINT_END. *event* is always NULL. *current_page* is the current page being printed. *across_pages* is the number of pages needed to span the width of the Outliner widget. *down_pages* is the number of pages needed to span the height of the Outliner widget. *total_pages* is the total number of pages to be printed (equal to *across_pages* × *down_pages*). *page_number* is the current printed page number, which is printed whenever the # symbol is specified for the page header or footer (by *XrtGearOutlinerVaDrawPS()*). *header* and *footer* are the current page header and footer. *fp* is the current output file pointer. *write_file_header* indicates whether the PostScript header is to be written out. (*write_file_header* is True when *current_page* is 1.) *write_file_trailer* indicates whether the PostScript trailer is to be written out; it is automatically set to True when *current_page* = *total_pages*. *doit* indicates whether to skip this page when printing; it has no effect on *current_page* or *total_pages*.

If *header* or *footer* is changed, you must free the existing header or footer and allocate a new String. Pages are printed in left-to-right, top-to-bottom order.

XmNxrtGearPSFontMapList **XrtGearPSFontMap *** **NULL** **CSG**

Defines a list of PostScript font to X font correspondences, with an optional point size. Each element in the list must be of type XrtGearPSFontMap:

```
typedef struct {
    String      font_name, tag;
    String      PS_font;
    double      point_size;
} XrtGearPSFontMap;
```

The X font can be represented by either its full font name (*font_name*) or it s XmNfontList tag (*tag*). *PS_font* is the PostScript font, and *point_size* is the point_size; if *point_size* is 0.0, it is inferred from the font. The last item in the list must have the *PS_font* member set to NULL.

XmNxrtGearRepaint **Boolean** **True** **CSG**

Specifies whether to redraw the widget (this also affects size recalculations). Drawing takes place when this resource is set to True.

XmNxrtGearResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies callbacks called when the widget is resized. Each routine is passed a pointer to an `XrtGearResizeCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearResizeCallbackStruct */
)
```

where `XrtGearResizeCallbackStruct` is defined as

```
typedef struct {
    XrtGearReason   reason;         /* read-only */
    Xevent          *event;         /* read-only */
    Dimension       width;
    Dimension       height;
} XrtGearResizeCallbackStruct;
```

reason is `XRTGEAR_REASON_RESIZE`. *event* is the XEvent that triggered the callback, or `NULL` if the callback is called programmatically. *width* is the new width. *height* is the new height.

XmNxrtGearScrollBarHoriz **Widget** ***dynamic*** **CSG**

Specifies the widget ID of the horizontal scrollbar.

XmNxrtGearScrollBarHorizDisplayPolicy **XrtGearDisplayPolicy** **XRTGEAR_DISPLAY_AS_NEEDED** **CSG**

Specifies the horizontal scrollbar display policy. Valid values are:

`XRTGEAR_DISPLAY_AS_NEEDED`	Show scrollbar only when needed
`XRTGEAR_DISPLAY_ALWAYS`	Always show scrollbar
`XRTGEAR_DISPLAY_NEVER`	Never show scrollbar

XmNxrtGearScrollBarVert **Widget** ***dynamic*** **CSG**

Specifies the widget ID of the vertical scrollbar.

XmNxrtGearScrollBarVertDisplayPolicy **XrtGearDisplayPolicy** **XRTGEAR_DISPLAY_AS_NEEDED** **CSG**

Specifies the vertical scrollbar display policy. Valid values are:

`XRTGEAR_DISPLAY_AS_NEEDED`	Show scrollbar only when needed
`XRTGEAR_DISPLAY_ALWAYS`	Always show scrollbar
`XRTGEAR_DISPLAY_NEVER`	Never show scrollbar

XmNxrtGearSelectionCallback **XtCallbackList** **NULL** **CSG**

Specifies the callbacks called when a node is selected or deselected. Each routine is passed a pointer to an XrtGearSelectCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearSelectCallbackStruct */
)
```

where XrtGearSelectCallbackStruct is defined as

```
typedef struct {
    XrtGearReason   reason;                 /* read-only */
    XEvent          *event;                 /* read-only */
    Widget          container;              /* read-only */
    Widget          node;
    XrtGearObject   selected_node_list;     /* read-only */
    Boolean         doit;
} XrtGearSelectCallbackStruct;
```

reason can be XRTGEAR_REASON_SELECT_BEGIN (selection about to begin), XRTGEAR_REASON_SELECT_END (selection complete), XRTGEAR_REASON_UNSELECT_BEGIN (deselection about to begin), or XRTGEAR_REASON_UNSELECT_END (deselection about to end). *event* is the XEvent that triggered the callback, or NULL if the callback is called programmatically. *container* is the parent Outliner widget. *node* is the node to be selected or deselected. *selected_node_list* is the selected node or list of nodes.

When *reason* is XRTGEAR_REASON_SELECT_BEGIN, you can set *doit* to False to suppress the selection operation. When *reason* is XRTGEAR_REASON_UNSELECT_BEGIN, you can only set *doit* to False (and consequently suppress the deselection operation) if XmNxrtGearSelectionPolicy is XRTGEAR_SELECTION_POLICY_RANGE or XRTGEAR_SELECTION_POLICY_MULTIPLE. By default, *doit* is True, which means that the selection or deselection takes place.

If XmNxrtGearSelectionPolicy is set to XRTGEAR_SELECTION_POLICY_SINGLE, the callbacks specified by XmNxrtGearSelectionCallback are called in the following order, with the following values as the *reason* member of the XrtGearSelectCallbackStruct structure:

XRTGEAR_REASON_SELECT_BEGIN	on the newly selected node
XRTGEAR_REASON_UNSELECT_BEGIN	on the previously selected node
XRTGEAR_REASON_UNSELECT_END	on the previously selected node
XRTGEAR_REASON_SELECT_END	on the newly selected node

XmNxrtGearSelectedNodeList **XrtGearObject** **NULL** **G**

Specifies a pointer to a list of the selected nodes.

XmNxrtGearSelectionPolicy **XrtGearSelectionPolicy** **_SELECTION_POLICY_SINGLE_AUTO_SELECT** **CSG**

Specifies the range of the permitted node selections. Valid values are:

XRTGEAR_SELECTION_POLICY_NONE	Disallowed mode: no node selection permitted. Any node may be dragged.

XRTGEAR_SELECTION_POLICY_SINGLE	Single mode: only one node can be selected at a time; selection is toggled (re-selecting a selected node deselects it). Only the selected node may be dragged.
XRTGEAR_SELECTION_POLICY_SINGLE_AUTO_SELECT	Single Auto mode: only one node can be selected at a time; selection is not toggled (re-selecting a selected node does not deselect it). Only the selected node may be dragged.
XRTGEAR_SELECTION_POLICY_RANGE	Range mode: multiple nodes can be selected, provided they are in a contiguous range. Only the selected nodes may be dragged.
XRTGEAR_SELECTION_POLICY_MULTIPLE	Multiple mode: any number of nodes, contiguous or not, can be selected at any given time. Only the selected nodes may be dragged.
XRTGEAR_SELECTION_POLICY_MULTIPLE_AUTO_UNSELECT	Multiple Auto mode: clear all selected nodes before performing a selection/unselection operation on the current node. Only the selected nodes may be dragged.

This resource is used in conjunction with the `XmNxrtGearSelectionRestrictions` resource to specify the node selection options available to the end-user.

If the value of this resource is changed, all nodes selected by the end-user are deselected.

XmNxrtGearSelectionRestrictions XrtGearSelectionRestrictions XRTGEAR_RESTRICT_NOTHING CSG

Specifies restrictions on what can be selected. Valid values are OR'ed combinations of the following:

XRTGEAR_RESTRICT_NOTHING	No restrictions
XRTGEAR_RESTRICT_ITEMS	Cannot select items
XRTGEAR_RESTRICT_FOLDERS	Cannot select folders
XRTGEAR_RESTRICT_MULTI_LEVEL	Can only select siblings of previously selected nodes
XRTGEAR_RESTRICT_ALL	Cannot select anything

This resource is used in conjunction with the `XmNxrtGearSelectionPolicy` resource to specify the node selection options available to the end-user.

If the value of this resource is changed, all nodes selected by the end-user are deselected.

XmNxrtGearShadowThickness Dimension 3 CSG

Specifies the thickness of the border drawn around the widget.

XmNxrtGearTextEditor Widget *dynamic* G

Specifies the widget ID for the text widget used for cell editing. The resource is read-only. The value of this resource does not depend on whether a cell is currently being edited. Once the outliner is realized, it should always return a valid text widget.

XmNxrtGearTextHorizClipPolicy XrtGearTextClipPolicy XRTGEAR_TRUNCATE_WITH_ARROW CSG

Specifies how text should be drawn if there is not enough space for the full text. Valid values are:

`XRTGEAR_TRUNCATE`	Truncates the text.
`XRTGEAR_TRUNCATE_WITH_ARROW`	Truncates the text, and draws a clip arrow toward the truncated text. In the case of a centered node clipped on both sides, draws a clip arrow on either side.
`XRTGEAR_COMPRESS_WITH_ELLIPSIS`	Display the first and last part of the string with ellipses in the middle to represent the missing text.

XmNxrtGearTreeData String NULL CSG

Specifies the name of a file containing the data to be loaded into a tree, plus optional information about the contents of the file. This resource is primarily for use with resource files and builder tools.

The following is the syntax for the `XmNxrtGearTreeData` resource string when loaded from a resource file:

filename indent separator open_escape closed_escape

All syntax elements except for *filename* are optional. If an element is omitted, no elements following it can be supplied.

filename is the name of the file containing the data; each line of the file represents one node.

indent is a string that specifies one indent level of the node on a particular line. If *indent* is `DEFAULT` or is not supplied, `\t` is assumed.

separator is a string that breaks the text into discrete sections, which can be used to display a multicolumn outline. If separator is `DEFAULT` or is not supplied, the comma character `,` is used.

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is `DEFAULT` or is not supplied, the open parenthesis character `(` is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is `DEFAULT` or is not supplied, the close parenthesis character `)` is assumed.

Setting `XmNxrtGearTreeData` to `NULL` destroys all existing nodes.

XmNwidth Dimension *determined at runtime* CSG

Specifies the width of the widget in pixels.

XmNxrtGearWidthChar char 'M' CSG

Specifies the character used to calculate character width with `XmNxrtNodeIndent` when `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_CHAR`. The font used to calculate character width is the first one in the list specified by `XmNfontList`. If this resource is set to '`\0`', the widest character in the font is used.

XmNxrtGearWidthPreferredCount **int** **40** **CSG**

Specifies the multiplier factor to use when calculating the preferred width of the displayed portion of the tree. This resource is only used when `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_COLUMN` or `XRTGEAR_SIZE_CHAR`.

If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_COLUMN`, the value of `XmNxrtGearWidthPreferredCount` is multiplied by the average width of the columns to yield the preferred width of the displayed portion of the tree.

If `XmNxrtGearWidthSizingPolicy` is set to `XRTGEAR_SIZE_CHAR`, the value of `XmNxrtGearWidthPreferredCount` is multiplied by the value of `XmNxrtGearWidthChar` (in the first font specified by `XmNfontList`) to yield the preferred width of the displayed portion of the tree.

XmNxrtGearWidthResizePolicy **XrtGearWidgetResizePolicy** **XRTGEAR_WIDGET_RESIZE_VARIABLE** **CSG**

Specifies how a widget should try to resize itself. Valid values are:

`XRTGEAR_WIDGET_RESIZE_VARIABLE`	Resize to the width determined by `XmNxrtGearWidthSizingPolicy`.
`XRTGEAR_WIDGET_RESIZE_GROW_ONLY`	Only resize if the widget grows.
`XRTGEAR_WIDGET_RESIZE_FIXED`	Don't resize unless a parent requests it.

XmNxrtGearWidthSizingPolicy **XrtGearSizingPolicy** **XRTGEAR_SIZE_CHAR** **CSG**

Specifies the manner in which the preferred width of the displayed portion of the tree is calculated. Valid values are:

`XRTGEAR_SIZE_PIXEL`	Use the value of `XmNxrtGearWidthPreferredCount` to specify the preferred width.
`XRTGEAR_SIZE_CHAR`	Multiply the value of `XmNxrtGearWidthChar` by the value of `XmNxrtGearWidthPreferredCount` to obtain the preferred width.
`XRTGEAR_SIZE_COLUMN`	Multiply the value of the average width of the columns by the value of `XmNxrtGearWidthPreferredCount` to obtain the preferred width.
`XRTGEAR_SIZE_VARIABLE`	Use the widths of the non-collapsed nodes to calculate the preferred width.

XmNxrtGearWidthVirtual **int** ***dynamic*** **G**

Specifies the virtual width of the widget in pixels. This is maintained for use by the horizontal scrollbar. If the width is greater than `ULONG_MAX`, this value will be set to `ULONG_MAX`.

7.13.2 XrtNodeObject Resources

XmNeditable **Boolean** **True** **CSG**

This resource controls whether or not users are allowed to edit node labels belonging to this node.

XmNxrtGearLabel **XrtGearObject** **NULL** **CSG**

Specifies the text to display in the node. This can be an XrtString object (which can contain a String or an XmString), or an XrtList object. If the text is specified as an XrtList object, this object must be a list of XrtStrings.

XmNxrtGearNodeStyle **Widget** **NULL** **CSG**

Specifies the node style. If this resource is set to an XrtString object, the node object searches for a NodeStyle object with a name that matches the value of this resource.

If no matching object is found, or this resource is set to `NULL`, the node object uses the first NodeStyle object in the list specified by the Outliner's `XmNxrtGearNodeStyleList` resource. If no node styles are defined, an Outliner-defined default node style is used.

XmNxrtGearSelected **Boolean** **False** **CSG**

Specifies whether this node is selected or not.

XmNuserData **XtPointer** **NULL** **CSG**

Provides a pointer to user-maintained data associated with a node. Note that the end-user is responsible for freeing any data placed here.

7.13.3 XrtNodeFolder Object Resources

XmNxrtGearFolderState **XrtGearFolderState** **XRTGEAR_FOLDERSTATE_OPEN_ALL** **CSG**

Specifies the current state of a folder node. Valid values are:

`XRTGEAR_FOLDERSTATE_CLOSED`	folder closed; no children visible
`XRTGEAR_FOLDERSTATE_OPEN_FOLDERS`	folder open; only folder children visible
`XRTGEAR_FOLDERSTATE_OPEN_ITEMS`	folder open; only item children visible
`XRTGEAR_FOLDERSTATE_OPEN_ALL`	folder open; all children visible

This value can be controlled programmatically or through user interaction. The Outliner widget's `XmNxrtGearFolderStateMask` resource controls which states are valid.

XmNxrtGearNodeChildList **XrtGearObject** **NULL** **G**

Specifies a pointer to a list of all children of this node.

7.13.4 XrtNodeStyle Object Resources

XmNbackground **Pixel** ***inherited*** **CSG**

Specifies the background color for (deselected) nodes of this style. If this resource is not set, the value of the Outliner widget's `XmNbackground` resource is used.

XmNxrtGearBackgroundSelected **Pixel** ***see description*** **CSG**

Specifies the background color for selected nodes. If this resource is not set, the complement of the existing background color is used.

XmNxrtGearFolderShortcutShow **Boolean** **True** **CSG**

Specifies whether shortcut buttons for children of folder nodes with this style are to be displayed. The shortcut buttons only appear if the Outliner's `XmNxrtGearNodeIndent` resource is set to a value large enough to provide sufficient space.

If set to `False`, the resource overrides the setting of `XmNxrtGearFolderShortcutShow` on any children of folder nodes with this style.

XmNfontList **XmFontList** ***inherited*** **CSG**

Specifies the font to use when drawing (deselected) nodes of this style.

XmNxrtGearFontListSelected **XmFontList** **NULL** **CSG**

Specifies the font to use when drawing selected nodes. If this resource is not set, the font specified by `XmNfontList` is used.

XmNforeground **Pixel** ***inherited*** **CSG**

Specifies the foreground color for (deselected) nodes of this style. If this resource is not set, the value of the Outliner widget's `XmNforeground` resource is used.

XmNxrtGearForegroundSelected **Pixel** ***see description*** **CSG**

Specifies the foreground color for selected nodes. If this resource is not set, the complement of the existing foreground color is used.

XmNheight **Dimension** ***dynamic*** **G**

Specifies the height of nodes of this style.

XmNxrtGearIconClosed **XrtGearIcon**  **CSG**

Specifies the icon used for a deselected NodeFolder object whose folder state is `XRTGEAR_FOLDERSTATE_CLOSED`.

XmNxrtGearIconClosedSelected **XrtGearIcon**  **CSG**

Specifies the icon used for a selected NodeFolder object whose folder state is `XRTGEAR_FOLDERSTATE_CLOSED`.

XmNxrtGearIconItem **XrtGearIcon**  **CSG**

Specifies the icon used for an unselected item.

XmNxrtGearIconItemSelected **XrtGearIcon**  **CSG**

Specifies the icon used for a selected item.

XmNxrtGearIconOpen **XrtGearIcon**  **CSG**

Specifies the icon used for a deselected NodeFolder object whose folder state is anything other than `XRTGEAR_FOLDERSTATE_CLOSED`.

XmNxrtGearIconOpenSelected **XrtGearIcon**  **CSG**

Specifies the icon used for a selected NodeFolder object whose folder state is anything other than `XRTGEAR_FOLDERSTATE_CLOSED`.

XmNxrtGearLayoutSpacing **Dimension** **5** **CSG**

Specifies the amount of space in pixels between the icon and the label string.

XmNxrtGearLineColor **Pixel** ***inherited from XmNforeground*** **CSG**

Specifies the color of the lines connecting the children of folder nodes of this style. This makes it possible to draw lines whose color is different from that of the shortcut buttons.

XmNxrtGearLineStyle **XrtGearLineStyle** **XRTGEAR_LINESTYLE_DOTTED** **CSG**

Specifies the style of the lines that connect the children of folder nodes of this style. Valid styles are:

`XRTGEAR_LINESTYLE_NONE`	Draw no lines
`XRTGEAR_LINESTYLE_DOTTED`	Draw dotted lines
`XRTGEAR_LINESTYLE_SOLID`	Draw solid lines

XmNxrtGearLineWidth **Dimension** **1** **CSG**

Specifies the thickness of the lines (if drawn) that connect the children of folder nodes of this style.

XmNxrtGearShortcutColor **Pixel** ***inherited*** **CSG**

Specifies the color of the shortcut button for all children of folder nodes of this style.

XmNxrtGearShortcutShow **Boolean** **True** **CSG**

Specifies whether shortcut buttons are to be displayed on children of the node. The shortcut buttons only appear if the Outliner's `XmNxrtGearNodeIndent` resource is set to a value large enough to provide sufficient space.

XmNxrtGearShortcutSize **Dimension** **9** **CSG**

Specifies the width and the height of the shortcut buttons in pixels.

7.13.5 XrtColumn Object Resources

XmNbackground **Pixel** ***inherited*** **CSG**

Specifies the background color of the label. If this resource is not set, the value of the Outliner widget's `XmNbackground` resource is used.

XmNeditable **Boolean** **True** **CSG**

This resource controls whether or not users are allowed to edit node labels in this column. If set to `False`, it overrides the editability settings of nodes and folders in the column.

XmNfontList **XmFontList** ***inherited*** **CSG**

Specifies the font used for the label when the column is not selected.

XmNxrtGearFontListSelected **XmFontList** **NULL** **CSG**

Specifies the font used for the label when the column is selected. If this resource is not set, the font specified by `XmNfontList` is used.

XmNforeground **Pixel** ***inherited*** **CSG**

Specifies the foreground color of the label. If this resource is not set, the value of the Outliner widget's `XmNforeground` resource is used.

XmNheight **Dimension** ***dynamic*** **CSG**

Specifies the height of the column in pixels.

XmNxrtGearLabel **XrtGearObject** **NULL** **CSG**

Specifies the text of the column label.

XmNxrtGearLabelAlignment **XrtGearAlignment** **XRTGEAR_ALIGN_CENTER** **CSG**

Specifies the alignment of the text within the column label. Valid values are:

`XRTGEAR_ALIGN_BEGINNING`	Align to start of label
`XRTGEAR_ALIGN_CENTER`	Align to center of label
`XRTGEAR_ALIGN_END`	Align to end of label

XmNxrtGearMarginLeft **Dimension** **2** **CSG**

Specifies the space in pixels between the left edge of the column and the start of column data.

XmNxrtGearMarginRight **Dimension** **2** **CSG**

Specifies the space in pixels between the end of column data and the right edge of the column.

XmNxrtGearNodeAlignment **XrtGearAlignment** **XRTGEAR_ALIGN_BEGINNING** **CSG**

Specifies the alignment of node text within a column. Valid values are:

`XRTGEAR_ALIGN_BEGINNING`	Align to start of column
`XRTGEAR_ALIGN_CENTER`	Align to center of label
`XRTGEAR_ALIGN_END`	Align to end of label

Note that this resource does not affect the appearance of the first column (which contains the hierarchy information).

XmNxrtGearResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies the list of callbacks called when the column is resized. Each routine is passed a pointer to an `XrtGearEnhancedSizeCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
```

```
        Widget      widget,
        XtPointer   client_data,
        XtPointer   call_data /* points to an
                                 XrtGearEnhancedSizeCallbackStruct */
    )
```

where XrtGearEnhancedSizeCallbackStruct is defined as

```
    typedef struct {
        XrtGearReason   reason;         /* read-only */
        Xevent          *event;         /* read-only */
        Dimension       width;
        Dimension       height;
        Dimension       old_width;      /* read-only */
        Dimension       old_height;     /* read-only */
        Boolean         doit;
    } XrtGearEnhancedSizeCallbackStruct;
```

reason is XRTGEAR_REASON_RESIZE_BEGIN or XRTGEAR_REASON_RESIZE_END. *event* is NULL. *width* is the new width, and *height* is the new height; these can be modified when *reason* is XRTGEAR_REASON_RESIZE_BEGIN. *old_width* and *old_height* are the width and height before resizing.

When *reason* is XRTGEAR_REASON_RESIZE_BEGIN, you can suppress the resize operation by setting *doit* to False.

Note that this callback structure is compatible with the XrtGearResizeCallbackStruct callback structure (used by the Outliner widget's XmNxrtGearResizeCallback resource). The XRTGEAR_REASON_RESIZE_END callback reason (used in the XrtGearEnhancedResizeCallbackStruct structure) is equivalent to the XRTGEAR_REASON_RESIZE callback reason used by the XmNxrtGearResizeCallbackStruct structure.

XmNxrtGearSelected **Boolean** **False** **CSG**

Specifies whether this column is selected or not.

XmNwidth **Dimension** **100** **CSG**

Specifies the width of the column in pixels.

XmNxrtGearWidthMaximum **Dimension** **10000** **CSG**

Specifies the maximum width to which the end-user is permitted to resize the column. This resource does not affect node resizings performed by the program.

XmNxrtGearWidthMinimum **Dimension** **10** **CSG**

Specifies the minimum width to which the end-user is permitted to resize the column. This resource does not affect node resizings performed by the program.

XmNxrtGearWidthPreferredCount **int** **10** **CSG**

Specifies a multiplier factor for the preferred column width in characters when XmNxrtGearWidthSizingPolicy is set to XRTGEAR_SIZE_CHAR.

XmNxrtGearWidthResizePolicy **XrtGearColumnResizePolicy** **XRTGEAR_COLUMN_RESIZE_ALL** **CSG**

Specifies how the end-user is allowed to resize the column. Valid values are:

XRTGEAR_COLUMN_RESIZE_ALL	column and label resizing permitted
XRTGEAR_COLUMN_RESIZE_LABELS_ONLY	only column label resizing permitted
XRTGEAR_COLUMN_RESIZE_NONE	no column resizing permitted

XmNxrtGearWidthSizingPolicy **XrtGearSizingPolicy** **XRTGEAR_SIZE_VARIABLE** **CSG**

Specifies the width sizing policy to be used when calculating the preferred column width. Valid sizing policies are:

XRTGEAR_SIZE_PIXEL	Use the value of XmNwidth to specify the preferred column width.
XRTGEAR_SIZE_CHAR	Use the value of XmNxrtGearWidthPreferredCount multiplied by the value of the parent Outliner widget's XmNxrtGearWidthChar resource.
XRTGEAR_SIZE_LABEL_VARIABLE	Use the preferred width of the column label to calculate the width of the column.
XRTGEAR_SIZE_VARIABLE	Use the width of the widest node label (in this column) to calculate the width of the column.

XmNx **Position** ***dynamic*** **G**

Specifies the X-offset of the beginning of the column, from the left of the Outliner widget. The outliner uses this offset to calculate where to draw the column labels and node labels.

7.13.6 Resources Inherited by Outliner

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNinsertPosition	Composite
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNuserData	XmManager
XmNwidth	Core
XmNx	Core
XmNy	Core

7.13.7 Resources Inherited by the Objects

Resource	Inherited From
XtNdestroyCallback	Object

7.14 Procedures and Methods Reference

7.14.1 Outliner Procedures and Methods

XmCreateXrtOutliner()

Creates an Outliner widget.

```
Widget
XmCreateXrtOutliner(
    Widget          parent,
    String          name,
    Arg             *arglist,
    int             argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmIsXrtOutliner()

Returns `True` if a widget is an Outliner widget.

```
Boolean
XmIsXrtOutliner(
    Widget          widget
    )
```

XrtGearCvtStringToXmString()

Converts a String to an XmString.

```
XmString
XrtGearCvtStringToXmString
    String          string
    )
```

XrtGearCvtXmStringToString()

Converts an XmString to a String. The String should be freed after use.

```
String
XrtGearCvtXmStringToString
    XmString        string
    )
```

XrtGearMrmInitialize()

Registers the XRT/gear widgets with Mrm, for use by UIL applications. This procedure must be called after *MrmInitialize()*.

```
void
XrtGearMrmInitialize(void)
```

XrtGearOutlinerCancelEdit()

Cancels interactive editing of a cell and unmaps the current editor. Any text entered by the user is discarded. The validate callback is not called.

XrtGearOutlinerCancelEdit() returns `False` if no edit is in progress or if any argument is invalid.

```
Boolean
XrtGearOutlinerCancelEdit(
    Widget      Outliner,
    )
```

outliner is a XRT/gear outliner widget.

XrtGearOutlinerCommitEdit()

Commits interactive cell editing to the outliner's node hierarchy, saving any chages made by the user. The method optionally unmaps the current editor. The callbacks in the `XmNxrtGearNodeValidateCallback` list are called first to verify that the changes are valid. *XrtGearOutlinerCommitEdit()* returns `False` if any argument is invalid.

```
Boolean
XrtGearOutlinerCommitEdit(
    Widget      Outliner,
    Boolean     unmap
    )
```

outliner is an XRT/gear outliner widget; *unmap* specifies whether to unmap the current editor if the method is successful.

XrtGearOutlinerDrawPS()

Generates encapsulated PostScript (EPSF-2.0) output of an Outliner tree, using PostScript fonts and device-independent PostScript operators. This method is identical to *XrtGearOutlinerVaDrawPS()*, but takes an arglist. *XrtGearOutlinerDrawPS()* returns the number of pages output (0 pages indicates an error). See *XrtGearOutlinerVaDrawPS()* on page 181 for a full description of the arguments.

```
int
XrtGearOutlinerDrawPS(
    Widget          outliner,
    FILE            *file,
    XrtGearPSarg    *arglist,
    int             argcount
    )
```

outliner is an Outliner widget; *file* should point to a file that is already open and ready for writing; *arglist* is a list of *argcount* arguments.

XrtGearOutlinerMoveColumns()

Moves columns within an Outliner widget.

```
Boolean
XrtGearOutlinerMoveColumns(
    Widget          outliner,
    int             destination,
    int             source,
    int             num_to_move
    )
```

outliner is an Outliner widget. *destination* specifies the destination column. *source* is the first column to be moved. *num_to_move* is the number of columns to be moved.

The moved columns are placed either before or after the destination column, as follows:

- If the columns to be moved are already immediately to the left of the destination column, the moved columns are placed to the right of the destination.
- In all other cases, the moved columns are placed to the left of the destination.

Note that 0 specifies the node hierarchy column; this column cannot be moved, and cannot be used as the destination column.

XrtGearOutlinerNodeFromXEvent()

Returns the node over which an XEvent occurred.

```
Widget
XrtGearOutlinerNodeFromXEvent(
    Widget              outliner,
    XEvent              *event,
    XrtGearLocation     *location
    )
```

outliner is an Outliner widget containing the node over which the event occurred. *event* is the detected XEvent.

location is the returned position of the event, relative to the node over which it occurred. *location* can have any of the following values:

`XRTGEAR_LOCATION_ON_NODE`	Event is over the node
`XRTGEAR_LOCATION_ON_SHORTCUT`	Event is on the node's shortcut button
`XRTGEAR_LOCATION_ON_LABEL`	Event is on a column label
`XRTGEAR_LOCATION_NOWHERE`	Event is not within any node

If *location* is `NULL`, no event location is returned.

XrtGearOutlinerNodeFromXY()

Returns the node corresponding to an XY position (relative to the root window).

```
Widget
XrtGearOutlinerNodeFromXY(
    Widget              outliner,
    int                 root_x,
    int                 root_y,
    XrtGearLocation     *location
    )
```

outliner is an Outliner widget containing the node in which the position is located; *root_x* is the X-coordinate of the XY position; *root_y* is the Y-coordinate.

location is the returned position, relative to the node over which it occurred. *location* can have any of the following values:

`XRTGEAR_LOCATION_ON_NODE`	Position is over the node
`XRTGEAR_LOCATION_ON_SHORTCUT`	Position is on the node's shortcut button
`XRTGEAR_LOCATION_ON_LABEL`	Position is on a column label

XRTGEAR_LOCATION_NOWHERE	Position is not within any node's sphere of influence

If *location* is NULL, no location is returned.

XrtGearOutlinerTraverseToCell()

Tries to move the current cell to a new location. *XrtGearOutlinerCommitEdit()* is first called to commit the edit in the current cell. If *XrtGearOutlinerCommitEdit()* returns False, *XrtGearOutlinerTraverseToCell()* returns False and does not move the XmText widget. If *XrtGearOutlinerCommitEdit()* returns True, *XrtGearOutlinerTraverseToCell()* tries to traverse to the specified cell. If the target cell is not editable, *XrtGearOutlinerTraverseToCell()* returns False. If XmNxrtGearNodeEnterCellCallback disallows the traversal, *XrtGearOutlinerTraverseToCell()* returns False and the current cell remains unchanged. Otherwise, the editor is mapped on top of the target cell.

If the cell is not the visible portion of the outliner, the outliner is scrolled until the cell is visible. If the cell is in a collapsed branch, the branch is opened recursively until the cell is visible. If XmNxrtGearFolderStateMask prevents this action, *XrtGearOutlinerTraverseToCell()* returns False and the XmText widget is not moved.

```
Boolean
XrtGearOutlinerTraverseToCell(
    Widget      outliner,
    Widget      node,
    int         column,
    )
```

outliner is the XRT/gear outliner; *node* is the node to traverse to; *column* is the column of the specified node to map the editor to.

XrtGearOutlinerVaDrawPS()

Generates encapsulated PostScript (EPSF-2.0) output of an Outliner tree, using PostScript fonts and device-independent PostScript operators. *XrtGearOutlinerVaDrawPS()* uses a *varargs*-style interface similar to *XtVaSetValues()*, using attribute-value pairs of type XrtGearDrawPSarg. *XrtGearOutlinerVaDrawPS()* returns the number of pages output (0 pages indicates an error).

```
int
XrtGearOutlinerVaDrawPS(
    Widget              outliner,
    FILE                *file,
    ...,
    )
```

outliner is an Outliner widget; *file* should point to a file that is already open and ready for writing. The following summarizes the attributes and values:

Attribute	Values/Type	Default
XRTGEAR_PS_BORDER_TYPE	XRTGEAR_BORDER_NONE XRTGEAR_BORDER_PLAIN XRTGEAR_BORDER_IN XRTGEAR_BORDER_OUT XRTGEAR_BORDER_ETCHED_IN XRTGEAR_BORDER_ETCHED_OUT XRTGEAR_BORDER_FRAME_IN XRTGEAR_BORDER_FRAME_OUT XRTGEAR_BORDER_BEVEL	inherited from `XmNxrtGearBorderType`
XRTGEAR_PS_COLOR	XRTGEAR_PS_COLOR XRTGEAR_PS_MONO	XRTGEAR_PS_COLOR
XRTGEAR_PS_COLUMN_PAGE_BREAK_LIST	XrtGearObject	NULL
XRTGEAR_PS_FOOTER	String	NULL
XRTGEAR_PS_FOOTER_FONT	String	"Courier "
XRTGEAR_PS_FOOTER_MARGIN	double	see description
XRTGEAR_PS_FOOTER_POINT_SIZE	double	12.0
XRTGEAR_PS_HEADER	String	NULL
XRTGEAR_PS_HEADER_FONT	String	"Courier "
XRTGEAR_PS_HEADER_MARGIN	double	see description
XRTGEAR_PS_HEADER_POINT_SIZE	double	12.0
XRTGEAR_PS_MARGIN_BOTTOM	double	see description
XRTGEAR_PS_MARGIN_LEFT	double	see description
XRTGEAR_PS_MARGIN_RIGHT	double	see description
XRTGEAR_PS_MARGIN_TOP	double	see description
XRTGEAR_PS_NODE_PAGE_BREAK_LIST	XrtGearObject	NULL
XRTGEAR_PS_NUM_COPIES	int	1
XRTGEAR_PS_ORIENTATION	XRTGEAR_PS_PORTRAIT XRTGEAR_PS_LANDSCAPE	XRTGEAR_PS_PORTRAIT
XRTGEAR_PS_PAPERSIZE_HEIGHT	double	11.0
XRTGEAR_PS_PAPERSIZE_WIDTH	double	8.5
XRTGEAR_PS_SCALE	double XRTGEAR_FIT_TO_PAGE XRTGEAR_FIT_TO_PAGE_WIDTH XRTGEAR_FIT_TO_PAGE_HEIGHT	72.0
XRTGEAR_PS_SHOW_COLUMN_LABELS	XRTGEAR_PS_EDGE_PAGES XRTGEAR_PS_ALL XRTGEAR_PS_NONE	XRTGEAR_PS_EDGE_PAGES

Attribute	Values/Type	Default
XRTGEAR_PS_SHOWPAGE	Boolean	True
XRTGEAR_PS_SHOW_START_NODE	Boolean	True
XRTGEAR_PS_START_NODE	Widget	NULL
XRTGEAR_PS_UNITS	XRTGEAR_PS_INCHES XRTGEAR_PS_CM	XRTGEAR_PS_INCHES

XRTGEAR_PS_BORDER_TYPE specifies the border type used when printing; this enables you to use different border types for display and printing. Valid values are the same as those for the Outliner's XmNxrtGearBorderType resource, discussed on page 126. The default is the border value specified by XmNxrtGearBorderType.

XRTGEAR_PS_COLOR specifies how colors are handled, and is either XRTGEAR_PS_COLOR (output in color) or XRTGEAR_PS_MONO (output in monochrome). When set to XRTGEAR_PS_MONO, all foreground colors are set to black, all background colors are set to white, and all shadows (if printed) are set to gray.

XRTGEAR_PS_COLUMN_PAGE_BREAK_LIST is an XrtList object whose elements are XrtColumn objects. Horizontal page breaks occur before each of these columns is printed. If no columns are specified (the default), page breaks are determined automatically.

XRTGEAR_PS_FOOTER specifies a string to be included as a footer on each page of output. Similarly, XRTGEAR_PS_HEADER specifies a string to be included as a header on each page of output.

XRTGEAR_PS_FOOTER_FONT and XRTGEAR_PS_HEADER_FONT specify the font for the header and footer, respectively. The default is Courier font.

XRTGEAR_PS_FOOTER_POINT_SIZE and XRTGEAR_PS_HEADER_POINT_SIZE specify the point size of the footer and header, if specified. The default value for each is 12.0 points.

XRTGEAR_PS_FOOTER, and XRTGEAR_PS_HEADER allow special characters to be embedded. By default, headers and footers are left-aligned, but can be center- or right-aligned by using the "|" character. The first occurrence of "|" centers the text which follows. A second occurrence right-justifies the text which follows. Other valid symbols:

#	current page number
^	total number of pages
%<x>	current date in *strftime()* format
\	next character is a literal

XRTGEAR_PS_FOOTER_MARGIN is only used if XRTGEAR_PS_FOOTER is set. It specifies the margin between the footer and the edge of the page. Its default, if a footer is specified, is 0.25 units (set by XRTGEAR_PS_UNITS).

XRTGEAR_PS_HEADER_MARGIN is only used if XRTGEAR_PS_HEADER is set. It specifies the margin between the header and the edge of the page. Its default, if a header is specified, is 0.25 units (set by XRTGEAR_PS_UNITS).

XRTGEAR_PS_MARGIN_TOP and XRTGEAR_PS_MARGIN_BOTTOM specify the width and the non-printing boundary arount the top and bottom edges of the paper (*not* around the top and bottom of the image). The default value for these attributes is 0.25 units (set by XRTGEAR_PS_UNITS); however:

- if XRTGEAR_PS_HEADER is set and XRTGEAR_PS_ORIENTATION is set to XRTGEAR_PS_PORTRAIT, the default value for XRTGEAR_PS_MARGIN_TOP is 0.75 units.
- if XRTGEAR_PS_FOOTER is set and XRTGEAR_PS_ORIENTATION is set to XRTGEAR_PS_PORTRAIT, the default value for XRTGEAR_PS_MARGIN_BOTTOM is 0.75 units.

XRTGEAR_PS_MARGIN_LEFT and XRTGEAR_PS_MARGIN_RIGHT specify the width and the non-printing boundary around the left and right edges of the paper (*not* around the left and right of the image). The default value for these attributes is 0.25 units (set by XRTGEAR_PS_UNITS); however:

- if XRTGEAR_PS_HEADER is set and XRTGEAR_PS_ORIENTATION is set to XRTGEAR_PS_LANDSCAPE, the default value for XRTGEAR_PS_MARGIN_LEFT is 0.75 units.
- if XRTGEAR_PS_FOOTER is set and XRTGEAR_PS_ORIENTATION is set to XRTGEAR_PS_LANDSCAPE, the default value for XRTGEAR_PS_MARGIN_RIGHT is 0.75 units.

XRTGEAR_PS_NODE_PAGE_BREAK_LIST is an XrtList object whose elements are XrtNode objects. Vertical page breaks occur before each of these nodes is printed. If no nodes are specified (the default), page breaks are determined automatically.

XRTGEAR_PS_NUM_COPIES specifies the number of copies to print. If the output is intended for inclusion in another document, this must be 1.

XRTGEAR_PS_ORIENTATION specifies the orientation of the image on the paper. Setting this attribute to XRTGEAR_PS_PORTRAIT specifies that the image is to be printed as it appears on the screen. Setting it to XRTGEAR_PS_LANDSCAPE rotates the Outliner widget 90 degrees counter-clockwise.

XRTGEAR_PS_PAPERSIZE_WIDTH and XRTGEAR_PS_PAPERSIZE_HEIGHT specify the width and height of the paper being used. The width and height do not change with the image orientation. Since paper always feeds through a printer oriented in the same way, regardless of how the image on the page is oriented, the paper width and height

attributes are tied to the top and left edges of the paper and not to the image. The number specified here is in units set by XRTGEAR_PS_UNITS. Common paper sizes:

- 8.5 x 11 inches (Letter)
- 8.5 x 14 inches (Legal)
- 21.0 x 29.7 centimeters (A4)
- 14.8 x 21.0 centimeters (A5)

XRTGEAR_PS_SCALE specifies the scaling of the widget image in pixels per inch. To approximate the size of the screen image, set this attribute to the resolution of the screen (the default value of 72.0 is the most common screen resolution). To increase the image size to 200%, set this attribute to 32.0; to decrease the image to 50%, set it to 144.0. The following enumerated values can be used to specify other scaling behaviour:

- XRTGEAR_FIT_TO_PAGE scales the image to fit on the page.
- XRTGEAR_FIT_TO_PAGE_WIDTH scales the image so it fits the width of the page.
- XRTGEAR_FIT_TO_PAGE_HEIGHT scales the image so it fits the height of the page.

XRTGEAR_PS_SHOW_COLUMN_LABELS specifies whether to include labels in the PostScript output. XRTGEAR_PS_ALL puts column labels on every page. XRTGEAR_PS_NONE removes column labels from all pages. XRTGEAR_PS_EDGE_PAGES includes labels only on the edge pages of multipage output, treating the output as a huge image broken into multiple pages (if there is only one page, the effect is the same as XRTGEAR_PS_ALL).

XRTGEAR_PS_SHOW_START_NODE only applies to folder nodes. If True (the default), the start node in the printed tree hierarchy is printed. This attribute has no effect if the start node is the Outliner widget.

XRTGEAR_PS_START_NODE specifies the first node in the tree hierarchy to print. If set to NULL (the default), printing starts with the root.

XRTGEAR_PS_UNITS specifies measurement units for the XRTGEAR_PAPERSIZE_ and XRTGEAR_MARGIN_ attributes.

7.14.2 XrtNode and XrtNodeFolder Procedures and Methods

XmCreateXrtNode()

Creates a node.

```
Widget
XmCreateXrtNode(
    Widget              parent,
    String              name,
    Arg                 *arglist,
    int                 argcount
    )
```

parent is the parent widget; *name* is the created node's name; *arglist* is a list of *argcount* arguments.

XmCreateXrtNodeFolder()

Creates a folder node.

```
Widget
XmCreateXrtNodeFolder(
    Widget              parent,
    String              name,
    Arg                 *arglist,
    int                 argcount
    )
```

parent is the parent widget; *name* is the created folder node's name; *arglist* is a list of *argcount* arguments.

XmIsXrtNode()

Returns `True` if its argument is a node.

```
Boolean
XmIsXrtNode(
    Widget              widget
    )
```

XmIsXrtNodeFolder()

Returns `True` if its argument is a folder node.

```
Boolean
XmIsXrtNodeFolder(
    Widget              widget
    )
```

XrtGearNodeChangeFolderState()

Sets the folder state. Callbacks specified by `XmNxrtGearFolderStateCallback` are called as the state is changed.

```
void
XrtGearNodeChangeFolderState(
    Widget              folder,
    XrtGearFolderState  state
    )
```

folder is the folder (regular or lightweight) whose state is being changed. *state* is the new state; see the description of the `XmNxrtGearFolderStateMask` resource for the valid values of this parameter.

XrtGearNodeCmpLabels()

A procedure which takes two nodes and compares their labels (useful for setting the Outliner's `XmNxrtGearNodeCompareProc` resource). Returns 1 if the first item is greater, 0 if they are equal, and -1 if the second item is greater.

```
int
XrtGearNodeCmpLabels(
    XtPointer           item1,
    XtPointer           item2
    )
```

item1 and *item2* are pointers to the nodes whose labels are to be compared.

XrtGearNodeCmpPositionsInTree()

Compares the position of two nodes in a tree. Returns a negative number if the first node appears before the second in the tree hierarchy, 0 if the two nodes are the same, or a positive number if the second node appears before the first.

```
int
XrtGearNodeCmpPositionsInTree(
    Widget              root,
    Widget              node1,
    Widget              node2
    )
```

root is the root node of the tree; *node1* and *node2* are the nodes to be compared.

XrtGearNodeConstructPath()

Finds the tree path leading from the root node of a tree to a specified node in the tree, and constructs a character string consisting of the labels of the nodes in this path. A pointer to the constructed string is returned.

```
String
XrtGearNodeConstructPath(
    Widget              root,
    Widget              node,
    Boolean             include_root,
    String              separator,
    String              label_separator,
    String              buffer
    )
```

root is the root node of the tree. *node* is the node to which the path is to be constructed. *include_root* indicates whether the name of the root node is to be included in the path; if *root* is the Outliner widget itself, the name of the widget (i.e., the name returned by *XtName()*) is included in the path. *separator* is the character string that separates pairs of nodes in the path; if *separator* is `NULL`, "/" is used. *label_separator* is the character string that separates multiple labels of a single node; if *label_separator* is `NULL`, " ," is used. If *buffer* is supplied (is non-`NULL`), it contains the constructed path name.

Note that if *buffer* is `NULL`, the path name returned by *XrtGearNodeConstructPath()* is internal memory, which can be overwritten at any time. To keep a copy of the path name, use the *buffer* argument.

XrtGearNodeCreateTreeFromBuffer()

Creates a tree or subtree from a multi-line text string (assumed to be in data file format). The return value depends on the value of *parent*, described below.

```
Widget
XrtGearNodeCreateTreeFromBuffer(
    Widget                  parent,
    char                    *buffer,
    Boolean                 full_path,
    XrtGearFolderState      initial_state,
    String                  indent,
    String                  separator,
    String                  open_escape,
    String                  close_escape,
    )
```

parent is the parent of the newly-created tree. *buffer* is the multi-line text string; each line represents a node of the created tree. Note: if this is not added under a valid widget, any type information contained in the buffer is lost.

The value returned by *XrtGearNodeCreateTreeFromBuffer()* depends on the value of *parent*. If *parent* is an Outliner widget or NodeFolder object, it is returned. Otherwise, a root NodeFolder object is created to contain the tree and is returned.

full_path indicates whether the complete tree hierarchy is provided for each node (*full_path* is `True`) or inferred from the indent levels (*full_path* is `False`). If *full_path* is `True`, the string specified by *indent* separates the tree hierarchy into its components. Creating a tree with *full_path* set to `True` can slow outliner. You should consider creating outliners with *full_path* set to `False`.

initial_state is the initial state of the folders in the tree.

indent is a string that specifies one indent level of the node on a particular line. If *indent* is `NULL`, "`\t`" (the tab character) is assumed.

separator is a string that breaks the text into discrete sections, which can be used to display a multicolumn outline. The default is "," (the comma character).

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is `NULL`, "`(`" is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is `NULL`, "`)`" is assumed.

XrtGearNodeCreateTreeFromStream()

Creates a tree from a data file or other input stream. The return value depends on the value of *parent*, described below.

```
Widget
XrtGearNodeCreateTreeFromStream(
    Widget                  parent,
    FILE                    *stream,
    Boolean                 full_path,
    XrtGearFolderState      initial_state,
    String                  indent,
    String                  separator,
    String                  open_escape,
    String                  close_escape,
    )
```

parent is the parent of the newly-created tree. *stream* is the data file; each line of the file represents a node of the tree. Note: if this is not added under a valid widget, any type information contained in the file is lost.

The value returned by *XrtGearNodeCreateTreeFromStream()* depends on the value of *parent*. If *parent* is an Outliner widget or NodeFolder object, it is returned. Otherwise, a root NodeFolder object is created to contain the tree and is returned.

full_path indicates whether the complete tree hierarchy is provided for each node (*full_path* is `True`) or inferred from the indent levels (*full_path* is `False`). If *full_path* is `True`, the string specified by *indent* separates the tree hierarchy into its

components.Creating a tree with *full_path* set to `True` can slow outliner. You should consider creating outliners with *full_path* set to `False`.

initial_state is the initial state of the folders in the tree.

indent is a string that specifies one indent level of the node on a particular line. If *indent* is `NULL`, "`\t`" (the tab character) is assumed.

separator is a string that breaks the text into discrete sections, which can be used to display a multicolumn outline. The default is "`,`" (the comma character).

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is `NULL`, "`(`" is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is `NULL`, "`)`" is assumed.

XrtGearNodeCvtLabelToString()

Returns the label of a specified node as a printable string. If the node's label is a list of strings, return a printable string consisting of all the strings in the list joined together, with each pair of strings in the list separated by a separator string. In either case, a pointer to the constructed label is returned.

```
String
XrtGearNodeCvtLabelToString(
    Widget          node,
    String          separator,
    String          buffer
    )
```

node is the node whose label is to be retrieved. *separator* is the separator string. If *buffer* is not `NULL`, it contains the constructed label.

Note that if *buffer* is `NULL`, the label returned by *XrtGearNodeCvtLabelToString()* is internal memory, which can be overwritten at any time. To keep a copy of the label, use the *buffer* argument.

XrtGearNodeCvtStringToLabel()

Converts a printable string to an XrtGearObject, suitable for use as a node label.

```
XrtGearObject
XrtGearNodeCvtStringToLabel(
    String          string,
    String          separator
    )
```

string is the string to be converted; its contents are unchanged. *separator* is the separator string; if present, it is used to separate the converted string into multiple labels.

If multiple labels are detected, the returned XrtGearObject is an XrtList object whose elements are XrtStrings. If the entire string specified by *string* is a single label, the returned XrtGearObject is an XrtString. In either case, the returned XrtGearObject can be used as the value of the `XmNxrtGearLabel` resource.

XrtGearNodeGetChild()

Returns the child corresponding to the specified index, or NULL if no such child exists.

```
Widget
XrtGearNodeGetChild(
    Widget      node,
    int         child_index
    )
```

node can be either an Outliner widget or a folder node object; *child_index* is the index of the child to be returned.

XrtGearNodeGetFirstInTree()

Returns the first child of the tree root, or NULL if no children exist.

```
Widget
XrtGearNodeGetFirstInTree(
    Widget      root
    )
```

root is the root node of the tree.

XrtGearNodeGetFirstNonCollapsedInTree()

Returns the first non-collapsed child of the tree root, or NULL if no non-collapsed children exist.

```
Widget
XrtGearNodeGetFirstNonCollapsedInTree(
    Widget      root
    )
```

root is the root node of the tree.

XrtGearNodeGetLastInTree()

Returns the node displayed at the bottom of the tree, or NULL if the root node has no children.

```
Widget
XrtGearNodeGetLastInTree(
    Widget      root
    )
```

root is the root node of the tree whose bottom node is being returned.

XrtGearNodeGetLastNonCollapsedInTree()

Returns the last non-collapsed child in a hierarchical list of nodes, or NULL if the root node has no children.

```
Widget
XrtGearNodeGetLastNonCollapsedInTree(
    Widget      root
    )
```

root is the root node of the tree whose last non-collapsed child is being returned.

XrtGearNodeGetLevel()

Returns the level (or generation) of the given node as compared to its root node. It returns NULL on failure.

```
int
XrtGearNodeGetLevel(
    Widget      root,
    Widget      node
    )
```

root is the root node of the tree; *node* is the node whose level is to be retrieved.

XrtGearNodeGetNextInTree()

Given a node, returns the next node in the hierarchical list.

```
Widget
XrtGearNodeGetNextInTree(
    Widget      root,
    Widget      node
    )
```

root is the root node of the tree; *node* is the node whose hierarchical successor is to be retrieved. If *node* is the last node in the tree, this procedure returns NULL. If *root* is NULL, *root* is inferred to be the last non-null node for which an XmNxrtGearNodeChildList resource is defined.

XrtGearNodeGetNextNonCollapsedInTree()

Given a node, returns the next non-collapsed node in the hierarchical list.

```
Widget
XrtGearNodeGetNextNonCollapsedInTree(
    Widget      root,
    Widget      node
    )
```

root is the root node of the tree; *node* is the node whose hierarchical non-collapsed successor is to be retrieved. If *node* is the last non-collapsed node in the tree, this procedure returns NULL. If *root* is NULL, *root* is inferred to be the last non-NULL node for which an XmNxrtGearNodeChildList resource is defined.

XrtGearNodeGetNumChildren()

Returns the number of children of a widget or object (as specified by the XmNxrtGearNodeChildList resource).

```
int
XrtGearNodeGetNumChildren(
    Widget      node
)
```

node is a widget or object whose number of children is to be obtained.

XrtGearNodeGetPreviousInTree()

Given a node, returns the preceding node in the hierarchical list.

```
Widget
XrtGearNodeGetPreviousInTree(
    Widget      root,
    Widget      node
    )
```

root is the root node of the tree; *node* is the node whose hierarchical predecessor is to be retrieved. If *node* is the first node in the tree, this procedure returns `NULL`. If *root* is `NULL`, *root* is inferred to be the last non-`NULL` node for which an `XmNxrtGearNodeChildList` resource is defined.

XrtGearNodeGetPreviousNonCollapsedInTree()

Given a node, returns the preceding non-collapsed node in the hierarchical list.

```
Widget
XrtGearNodeGetPreviousNonCollapsedInTree(
    Widget      root,
    Widget      node
    )
```

root is the root node of the tree; *node* is the node whose hierarchical non-collapsed predecessor is to be retrieved. If *node* is the first non-collapsed node in the tree, this procedure returns `NULL`. If *root* is `NULL`, *root* is inferred to be the last non-`NULL` node for which an `XmNxrtGearNodeChildList` resource is defined.

XrtGearNodeGetVisibility()

Returns the current visibility status of a node.

```
XmVisibility
XrtGearNodeGetVisibility(
    Widget      node
    )
```

node is the node being checked. The returned XmVisibility value is one of the following:

```
XmVISIBILITY_UNOBSCURED
XmVISIBILITY_PARTIALLY_OBSCURED
XmVISIBILITY_FULLY_OBSCURED
```

XrtGearNodeGetWidgetParent()

Travels up the node hierarchy until a widget is reached, and returns it.

```
Widget
XrtGearNodeGetWidgetParent(
    Widget      node
    )
```

node is the node whose parent widget is to be retrieved.

XrtGearNodeIsAncestor()

Returns `True` if one specified node is an ancestor of another (i.e. is in the ancestor's subtree).

```
Boolean
XrtGearNodeIsAncestor(
    Widget      root,
    Widget      ancestor,
    Widget      node
    )
```

root is the root of the tree; if *root* is `NULL`, a root is inferred. *ancestor* is the specified ancestor node. *node* is the specified descendant.

XrtGearNodeIsInCollapsedBranch()

Returns `True` if the specified node is in a collapsed branch (if it is in the subtree of a closed folder).

```
Boolean
XrtGearNodeIsInCollapsedBranch(
    Widget      root,
    Widget      node
    )
```

root is the root of the tree; if *root* is `NULL`, a root is inferred. *node* is the node being checked.

XrtGearNodeMakeVisible()

Makes the specified node visible on the screen.

```
void
XrtGearNodeMakeVisible(
    Widget      node
    )
```

node is the node to be made visible. The node is unaffected if it is already visible.

XrtGearNodeSetParent()

Changes the parent of a node. Returns `True` if the reparenting is successful, or `False` if the reparenting fails for any reason.

```
Boolean
XrtGearNodeSetParent(
    Widget      node,
    Widget      new_parent,
    Widget      sibling
    )
```

node is the node to be reparented; *new_parent* is its new parent. *new_parent* must be a NodeFolder. If *sibling* is not `NULL`, *node* is placed immediately ahead of *sibling* in the list of children; in this case, *sibling* must be a child of *new_parent*. If *sibling* is `NULL` and *new_parent* has other children, *node* is added to the end of the list of children. (If `XmNxrtGearAutoSort` is `True`, *node* is placed in sorted order in the list of children, regardless of the value of *sibling*.)

XrtGearNodeSetValuesOnTree()

Sets the provided arg list on all children of the root node provided and optionally on the root node. root can also be a lightweight node.

```
void
XrtGearNodeSetValuesOnTree(
    Widget      root,
    Arg         *args,
    Cardinal    num_args,
    Boolean     Set_on_root
    )
```

XrtGearNodeSortTree()

Sorts the tree. Each sibling group is sorted separately.

```
Boolean
XrtGearNodeSortTree(
    Widget          root
    )
```

root is the root of the tree to be sorted.

XrtGearNodeTraverseTo()

Sets the `XmNxrtGearNodeCurrent` and forces the focus to the specified node. If the specified node is not visible, outliner will scroll to it. If the node is not open, it will only scroll to its visible parent. *XrtGearNodeTraverseTo()* does not select the node, however. It only gives it the focus.

```
void
XrtGearNodeTraverseTo(
    Widget          node
    )
```

node is the node which is to gain focus.

XrtGearNodeWriteTreeToBuffer()

Writes a tree or subtree into a buffer as a multi-line text string. An internal buffer containing the text string is returned. (This internal memory may be overwritten at any time. To preserve a copy of the text string, provide a pointer to a buffer as described below.)

```
String
XrtGearNodeWriteTreeToBuffer(
    Widget          parent,
    char            *buffer,
    Boolean         full_path,
    Boolean         write_root,
    String          indent,
    String          separator,
    String          open_escape,
    String          close_escape,
    int             *length
    )
```

parent is the parent of the tree to be written. *buffer*, if not `NULL`, is the text string into which the tree is to be written; each line of the text string represents a node of the

tree. If *buffer* is NULL, the tree is written into internal memory only; you should use it immediately or copy it.

full_path indicates whether the complete tree hierarchy is provided for each node (*full_path* is True) or whether only indent levels and the node label are provided (*full_path* is False). If *full_path* is True, the string specified by *indent* separates the tree hierarchy into its components.

write_root indicates whether the parent node is to be written to the buffer. If *write_root* is the Outliner widget, the name of the widget as returned by *XtName()* is written to the buffer.

indent is a string that specifies one indent level of the node on a particular line. If *indent* is NULL, "\t" (the tab character) is assumed.

separator is a string that separates multiple labels for a single node. The default is "," (the comma character).

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is NULL, "(" is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is NULL, ")" is assumed.

length, if not NULL, returns the length of the text string in bytes.

XrtGearNodeWriteTreeToStream()

Writes a tree to a data file. Returns True if the file was written successfully, or False if an error occurred.

```
Boolean
XrtGearNodeWriteTreeToStream(
    Widget              parent,
    FILE                *fp,
    Boolean             full_path,
    Boolean             write_root,
    String              indent,
    String              separator,
    String              open_escape,
    String              close_escape,
    int                 *length
    )
```

parent is the parent of the tree to be written. *fp* is the data file to which the tree is to be written; each line of the file represents a node of the tree.

full_path indicates whether the complete tree hierarchy is provided for each node (*full_path* is True) or whether only indent levels and the node label are provided (*full_path* is False). If *full_path* is True, the string specified by *indent* separates the tree hierarchy into its components.

write_root indicates whether the parent node is to be included in the data file. If *write_root* is the Outliner widget, the name of the widget as returned by *XtName()* is written to the file.

indent is a string that specifies one indent level of the node on a particular line. If *indent* is NULL, "\t" (the tab character) is assumed.

separator is a string that separates multiple labels for a single node. The default is "," (the comma character).

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is `NULL`, "`(`" is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is `NULL`, "`)`" is assumed.

length, if not `NULL`, returns the length of the written file in bytes.

7.14.3 XrtNodeStyle Procedures and Methods

XmCreateXrtNodeStyle()

Creates a NodeStyle.

```
Widget
XmCreateXrtNodeStyle(
    Widget      parent,
    String      name,
    Arg         *arglist,
    int         argcount
    )
```

parent is the parent; *name* is the created node style's name; *arglist* is a list of *argcount* arguments.

XmIsXrtNodeStyle()

Returns `True` if its argument is a NodeStyle.

```
Boolean
XmIsXrtNodeStyle(
    Widget      widget
    )
```

XrtGearNodeStyleFromName()

Searches an Outliner widget's node style list for a style matching a specified name.

```
Widget
XrtGearNodeStyleFromName(
    Widget      outliner,
    String      name
    )
```

outliner is an Outliner widget whose node style list is to be searched; *name* is the node style name to search for. If the node style is not found in the widget's style list, the routine returns `NULL`.

7.14.4 XrtColumn Procedures and Methods

XmCreateXrtColumn()

Creates a column.

```
Widget
XmCreateXrtColumn(
    Widget      parent,
    String      name,
    Arg         *arglist,
    int         argcount
    )
```

parent is the parent; *name* is the created column's name; *arglist* is a list of *argcount* arguments.

XmIsXrtColumn()

Returns `True` if its argument is a column.

```
Boolean
XmIsXrtColumn(
    Widget      widget
    )
```

XrtGearColumnFromXEvent()

Returns the column in which an XEvent occurred. This method returns `NULL` if the event did not occur over a column.

```
Widget
XrtGearColumnFromXEvent(
    Widget      outliner,
    XEvent      *event
    )
```

outliner is an Outliner widget; *event* is the event that has occurred.

XrtGearColumnFromXY()

Returns the column in which an X-Y position (relative to the root window) is located. This method returns `NULL` if the position does not correspond to a column.

```
Widget
XrtGearColumnFromXY(
    Widget      outliner,
    int         root_x,
    int         root_y,
    )
```

outliner is an Outliner widget; *root_x* is the X position. *root_y* is the Y position.

7.15 Translations and Actions

7.15.1 Translations for the Node Area Clip Child

The translations in use for the node area clip child depend on the value of `XmNxrtGearSelectionPolicy`. The following tables list the translations for each of the valid values of this resource. The default translations for the node area clip child are also listed in a table.

Single Auto Translations

These translations are used when selection is in Single Auto mode (`XmNxrtGearSelectionPolicy` is set to `XRTGEAR_SELECTION_POLICY_SINGLE_AUTO_SELECT`).

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	NoAction()
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) Edit(Pointer) GoTo(Pointer) ChangeState(ShortCut) Select(Start) ChangeState(Node,2) Activate(Node,2)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	GoTo(Pointer) Select(Start)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	GoTo(Pointer) Select(Start)
~Alt ~Ctrl ~Meta <Btn1Motion>	ResizeColumn(Extend)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	ResizeColumn(Cancel) CancelEdit()
Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection, Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Selection, Copy)
~Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection)
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage) Select(Current)

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage) Select(Current)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next) Select(Current)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous) Select(Current)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top) Select(Current)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom) Select(Current)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage) Select(Current)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage) Select(Current)
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next) Select(Current)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous) Select(Current)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top) Select(Current)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom) Select(Current)

Single Translations

These translations are used when selection is in Single mode (`XmNxrtGearSelectionPolicy` is set to `XRTGEAR_SELECTION_POLICY_SINGLE`).

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	Select(Start)
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) GoTo(Pointer) Edit(Pointer) ChangeState(ShortCut) Select(Start) ChangeState(Node,2) Activate(Node,2)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	GoTo(Pointer) Select(Start)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	GoTo(Pointer) Select(Start)
~Alt ~Ctrl ~Meta <Btn1Motion>	ResizeColumn(Extend)

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection, Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Selection, Copy)
~Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection)
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	Select(Cancel) ResizeColumn(Cancel) CancelEdit()
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom)

Range Translations

These translations are used when selection is in Range mode (`XmNxrtGearSelectionPolicy` is set to `XRTGEAR_SELECTION_POLICY_RANGE`).

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	Select(Start)
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) GoTo(Pointer) Edit(Pointer) ChangeState(ShortCut) Select(Start) ChangeState(Node,2) Activate(Node,2)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	GoTo(Pointer) Select(Extend)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	NoAction()
~Alt ~Ctrl ~Meta <Btn1Motion>	GoTo(Pointer) ResizeColumn(Extend) Select(Extend)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection, Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Selection, Copy)
~Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection)
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	Select(Cancel) ResizeColumn(Cancel) CancelEdit()
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom)

Event	Action or Actions
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom) Select(CurrentExtend)

Multiple Translations

These translations are used when selection is in Multiple mode (`XmNxrtGearSelectionPolicy` is set to `XRTGEAR_SELECTION_POLICY_MULTIPLE`).

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	Select(Add)
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) GoTo(Pointer) Edit(Pointer) ChangeState(ShortCut) Select(Add) ChangeState(Node,2) Activate(Node,2)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	GoTo(Pointer) Select(Extend)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	GoTo(Pointer) Select(Add)
~Alt ~Ctrl ~Meta <Btn1Motion>	GoTo(Pointer) ResizeColumn(Extend) Select(Extend)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection, Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Selection, Copy)
~Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection)
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)

Event	Action or Actions
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	Select(Cancel) ResizeColumn(Cancel) CancelEdit()
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom) Select(CurrentExtend)

Multiple Auto Translations

These translations are used when selection is in Multiple Auto mode (`XmNxrtGearSelectionPolicy` is set to `XRTGEAR_SELECTION_POLICY_MULTIPLE_AUTO_UNSELECT`).

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	Select(Start)

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) GoTo(Pointer) Edit(Pointer) ChangeState(ShortCut) Select(Start) ChangeState(Node,2) Activate(Node,2)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	GoTo(Pointer) Select(Extend)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	GoTo(Pointer) Select(Add)
~Alt ~Ctrl ~Meta <Btn1Motion>	GoTo(Pointer) ResizeColumn(Extend) Select(Extend)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection, Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Selection, Copy)
~Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Selection)
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	Select(Cancel) ResizeColumn(Cancel) CancelEdit()
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage) Select(CurrentExtend)

Event	Action or Actions
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top) Select(CurrentExtend)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom) Select(CurrentExtend)

Selection Disallowed Translations

These translations are used when selection is in Disallowed mode (`XmNxrtGearSelectionPolicy` is set to `XRTGEAR_SELECTION_POLICY_NONE`).

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	NoAction()
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) GoTo(Pointer) Edit(Pointer) ChangeState(ShortCut) ChangeState(Node,2) Activate(Node,2)
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	GoTo(Pointer)
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	GoTo(Pointer)
~Alt ~Ctrl ~Meta <Btn1Motion>	ResizeColumn(Extend)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Node, Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Node, Copy)
~Shift ~Alt ~Ctrl ~Meta <Btn2Down>	Drag(Node)
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	ResizeColumn(Cancel) CancelEdit()

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom)

Default Node Translations

Translations shown here are overridden by any of the translations defined above.

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Key>space	ChangeState(Current)Activate(Current)
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	NoAction()
Shift ~Alt ~Ctrl ~Meta <Btn1Down>	NoAction()
Ctrl ~Alt ~Meta ~Shift <Btn1Down>	NoAction()
~Alt ~Ctrl ~Meta <Btn1Motion>	NoAction()

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	NoAction()
~Alt ~Ctrl ~Meta ~Shift <Key>osfSelect	ChangeState(Current) Activate(Current)
<Key>osfActivate	ChangeState(Current) Activate(Current)
<Key>Return	ChangeState(Current) Activate(Current)
<Key>osfCancel	Select(Cancel) ResizeColumn(Cancel)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta ~Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta ~Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta ~Shift <Key>osfEndLine	GoTo(Bottom)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage)
~Alt ~Ctrl ~Meta Shift <Key>osfDown	GoTo(Next)
~Alt ~Ctrl ~Meta Shift <Key>osfUp	GoTo(Previous)
~Alt ~Ctrl ~Meta Shift <Key>osfBeginLine	GoTo(Top)
~Alt ~Ctrl ~Meta Shift <Key>osfEndLine	GoTo(Bottom)
Shift ~Ctrl ~Alt ~Meta <Btn2Down>	Drag(Move)
Ctrl ~Shift ~Alt ~Meta <Btn2Down>	Drag(Copy)
~Ctrl ~Alt ~Shift ~Meta <Btn2Down>	Drag(Move)
<Key>osfDelete	Delete()
<Key>osfHelp	PrimitiveHelp()
Shift ~Meta ~Alt <Key>Tab	PrimitivePrevTabGroup()

Event	Action or Actions
~Meta ~Alt <Key>Tab	PrimitiveNextTabGroup
<FocusIn>	PrimitiveFocusIn()
<FocusOut>	PrimitiveFocusOut()
<Unmap>	PrimitiveUnmap()

7.15.2 Translations for the Column Labels Clip Child

Event	Action or Actions
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>	ResizeColumn(Start) Select(Column)
~Alt ~Ctrl ~Meta ~Shift <Btn1Down>(2+)	Activate(Column)
~Alt ~Ctrl ~Meta ~Shift <Btn1Up>	ResizeColumn(End)
~Alt ~Meta <Btn2Down>	Drag(Column)

7.15.3 Translations for the XmText widget used for editing:

Event	Action or Actions
<Key>osfCancel	CancelEdit()
<Key>osfActivate	CommitEdit(True)
<Key>Return	CommitEdit(True)
~Alt ~Ctrl ~Meta Shift <Key>Tab	Edit(Previous)
~Alt ~Ctrl ~Meta ~Shift <Key>Tab	Edit(Next)
~Alt ~Meta <Key>osfPageDown	GoTo(NextPage) Edit(Current) Select(Start)
~Alt ~Meta <Key>osfPageUp	GoTo(PreviousPage) Edit(Current) Select(Start)
~Alt ~Meta <Key>osfDown	GoTo(Next) Edit(Current) Select(Start)
~Alt ~Meta <Key>osfUp	GoTo(Previous) Edit(Current) Select(Start)
~Alt ~Ctrl ~Meta Shift <Key>osfPageDown	GoTo(NextPage) Edit(Current) Select(Start)
~Alt ~Ctrl ~Meta Shift <Key>osfPageUp	GoTo(PreviousPage) Edit(Current) Select(Start)
~Alt Ctrl ~Meta <Key>osfLeft	Edit(Previous)
~Alt Ctrl ~Meta <Key>osfRight	Edit(Next)

7.15.4 Actions

Activate()

Calls the activate callback and is for application-defined use. It takes two parameters. The first parameter can be any of the following:

Node	Call the activate callbacks if the click occurs on the node (which includes the folder icon if one is present)
ShortCut	Call the activate callbacks if the shortcut button is clicked (the +/- box).
Current	Call the activate callbacks on the current node. This is for use with keyboard translations.
Column	Call the activate callbacks if a column label is clicked on.

The default behavior is for a double click on the node. The default activate keys are `space` and `osfActivate`.

The second parameter is optional, and specifies the number of clicks required to call the activate callbacks. The default is 1 (single click).

CancelEdit()

This action calls *XrtGearOutlinerCancelEdit()*. It is bound to the `osfCancel` key.

ChangeState()

Lets the programmer specify the conditions under which a state change occurs. It takes two parameters. The first parameter can be any of the following:

Node	Change state if the click(s) occur on the node (which includes the folder icon if one is present).
ShortCut	Change state if the shortcut button is clicked (the +/- box).
Current	Change the state on the current node. This is for use with keyboard translations.

The second parameter is optional, and specifies the number of clicks required to activate the state change.

The default behavior is the following:

- a double click to change state when clicking on a node
- a single click to change state if the click occurs over a shortcut button

The default state-changing keys are `space` and `osfActivate`.

CommitEdit()

Since *XrtGearOutlinerCommitEdit()* cannot be used, as the data going to the validate callback is different for a user-initiated commit, this action is used. It takes one parameter of either `True` or `False`, indicating whether or not to unmap the XmText widget.

This action is bound to the `osfActivate` and the `Return` keys.

Delete()

Deletes the current node. This calls *XtDestroyWidget()* to remove the node. If you need to be informed of the destruction of any particular node, add a destroy callback routine when creating the node, using the `XmNdestroyCallback` resource. For registering nodes that are added by the Outliner widget itself, add the destroy callback to the list specified by `XmNxrtGearNodeCreateCallback`.

By default, the *Delete()* action is bound to `osfDelete`.

Drag()

Allows the user to interactively drag a node to a new location. This action must be used in conjunction with a Mouse event. This action is passed two parameters. The first specifies the item(s) to be dragged, and is one of the following:

Node	Drag the current node.
Selection	Drag the currently selected items.
Column	Drag the currently selected column.

The second argument defines the action, and can be one of:

Move	Move the specified items to the new location
Copy	Insert a copy of the items at the new location

If the second argument is not specified, Move is assumed. If the first argument is Column, the second argument cannot be Copy.

Note that when a node is copied a new one is created, and the callbacks specified by `XmNxrtGearNodeCreateCallback` are invoked.

Edit()

Maps the editor widget to the specified location. The parameter can be one of:

Pointer	Edits the node that the mouse pointer is over.
Current	Edits the current node.
Next	Edits the next sibling of the current node.
Previous	Edits the previous sibling of the current node.

GoTo()

Lets you move the focus through keystrokes and mouse actions to the position you specify. This action takes one parameter, which can be one of:

Pointer	Move focus to the node that the mouse pointer is over
Current	Make current node visible (if not already visible)
TopOfPage	Move focus to the top of the current page
BottomOfPage	Move focus to the bottom of the current page
NextPage	Move focus and display one page down
PreviousPage	Move focus and display one page up

ParentNode	Move focus to the parent
NextParent	Move focus to the node's parent's next sibling
Next	Move focus to the node's next sibling
Previous	Move focus to the node's previous sibling
TopOfTree	Move focus to the first child of the root node
BottomOfTree	Move focus to the last visible (grand)child of the root node

NoAction()

Does nothing, triggering no callbacks. By default, this action routine is not bound to any event. Setting an event to call this routine effectively disables the translation.

ResizeColumn()

If the cursor is positioned over the column border or column label border, this action routine begins and ends the resize of a column. It is passed one parameter, which is one of the following:

Start	Begins the resize of a column. The widget draws a dashed line that follows the mouse cursor. Drawing continues until *ResizeColumn()* is called with the `End` parameter.
Extend	Extend the dashed line to continue following the mouse cursor.
End	Call the callbacks specified by `XmNxrtGearResizeCallback`, and resize the column.

If `XmNxrtGearShadowThickness` is 0, columns cannot be interactively resized. If a column has its `XmNwidth` resource set to 0, it is hidden, and cannot be interactively resized.

Select()

Toggles the selection of a node: selected nodes become deselected, and vice versa. It requires one parameter, which can be one of:

Start	The node that has focus is to be selected. This unselects all other nodes.
Extend	Select from the last selected node to the current node.
CurrentExtend	Same as Extend; used by the traversal keys.
Add	Start a new selection without clearing the old selection.
Column	Selects the current column. This sets the `XmNxrtGearColumnSelected` resource.
Cancel	Unselect all selected nodes (unless `XmNxrtGearSelectionPolicy` is `XRTGEAR_SELECTION_POLICY_SINGLE_AUTO_SELECT`).

7.16 Lightweight Folder and Item Nodes

Regular folder and item nodes in Outliner are individual XtObjects. When creating very large Outliner applications, involving thousands of folder and item nodes, a large number of XtObjects can affect performance. To address this, Outliner provides "lightweight" folder and item nodes. Lightweight nodes are non-XtObject nodes that can be added to an Outliner tree. Lightweight nodes are compatible with their XtObject variants. If you so choose, you can mix regular and lightweight nodes in an Outliner application.

Building a Tree Using Lightweight Nodes

The example below demonstrates manually creating a tree and populating it with lightweight folders and items:

```
#include <stdio.h>
#include <stdlib.h>
#include <Xm/XrtGear.h>
#include <Xm/XrtNode.h>
#include <Xm/XrtLWNode.h>
#include <Xm/XrtLWFolder.h>
#include <Xm/XrtNodeFolder.h>
#include <Xm/XrtList.h>
#include <Xm/XrtGearString.h>
#include <Xm/XrtColumn.h>
#include <Xm/XrtOutliner.h>
#include <X11/Intrinsic.h>

int
main(int argc, char **argv)
{
     Widget toplevel, outliner;
     XtAppContext app_context;
     Widget node, folder;

     toplevel = XtAppInitialize(&app_context, "Sample", NULL,
      0, &argc, argv, NULL, NULL, 0);

     outliner = XtVaCreateManagedWidget("outliner",
      xmXrtOutlinerWidgetClass, toplevel, NULL);

     folder = XrtGearFolderCreateLW(outliner, "Folder",
      XrtGearStringCreateCharString("Folder"),
      XRTGEAR_FOLDERSTATE_OPEN_ALL);

     node = XrtGearNodeCreateLW(folder, "Node",
      XrtGearStringCreateCharString("item1"));

     node = XrtGearNodeCreateLW(folder, "Node",
      XrtGearStringCreateCharString("item2"));

     XtRealizeWidget(toplevel);
     XtAppMainLoop(app_context);
}
```

7.16.1 Lightweight Nodes Cast as Widgets

As noted previously, lightweight nodes are not XtObjects. However, methods that create lightweight nodes (for example, *XrtGearNodeCreateLW()* and *XrtGearFolderCreateLW()*) return lightweight nodes cast as Widgets for compatibility with existing code.

If `XtIsObject()` or `XtIsWidget()` is passed a lightweight node, it will return `False`.

Note, `XtIsObject()` and `XtIsWidget() are` the only Xt macros/procedures that can safely be passed a lightweight node. You must use the procedures supplied by XRT/gear on a lightweight node. For example, use *XrtGearNodeGetParent()* instead of *XtParent()*. However, XtObject nodes can be passed to XRT/gear node functions. For example, if *XrtGearNodeGetParent()* is passed an XrtObject node, it calls *XtParent()* to get the parent of the node.

7.16.2 Getting and Setting Lightweight Node Resources

Lightweight node are non-XtObjects, therefore you cannot manipulate properties with *XtVaGetValues()* and *XtVaSetValues()*. By including *XrtLWFolder.h* and *XrtLWNode.h*, Outliner provides an API that allows you to create and destroy Lightweight nodes, set and get resources (most resources are retrieved and set with individual methods), and perform other functions.

The following code sets a lightweight item node's `Style` resources:

```
   Widget outliner, nodestyle;
   Widget node, folder;

   nodestyle = XmCreateXrtNodeStyle(outliner,
       "mystyle", NULL, NULL);

/* XtVaSetValues() used to set XtObject resources */

   XtVaSetValues(nodestyle,
       XtVaTypedArg,   XmNxrtGearBackgroundSelected,
       XmRString,      "white", strlen("white")+1,
       NULL);
   XtVaSetValues(nodestyle,
       XtVaTypedArg,   XmNxrtGearFontListSelected,
       XmRString,      "-*-terminal-bold-r-*-*-14-*-*-*-*_*-iso8859-*",
       strlen("-*-terminal-bold-r-*-*-14-*-*-*-*_*-iso8859-*")+1,
       NULL);

   node = XrtGearNodeCreateLW(folder, "Node",
         XrtGearStringCreateCharString("item1"));

/* XrtGearNodeSetNodeStyle() used to set lightweight
   node resources */

   XrtGearNodeSetNodeStyle(node, nodestyle);
```

7.16.3 Destroying Lightweight Nodes

You cannot use *XtDestroyWidget()* to destroy lightweight item and folder nodes. To destroy a lightweight node, use *XrtGearNodeDestroy()* instead. For example:

```
XrtGearNodeDestroy(node);
XrtGearNodeDestroy(folder);
```

7.16.4 Lightweight Folder Demo

The fileviewer demo has been modified to optionally include lightweight nodes. While the pre-built demo uses regular nodes, the source code for the new fileviewer demo uses both lightweight nodes and regular nodes. You can compile either a regular node or lightweight node version using the embedded #ifdefs.

```
#ifdef XRTGEAR_LW
...
#else
...
#endif
```

To build the lightweight node version of the fileviewer demo, define the `XRTGEAR_LW` symbol for the *make* process. Note that this step is platform dependent. For example on SunOS you can use:

```
make all "CC= cc -DXRTGEAR_LW"
```

7.16.5 Lightweight Item and Folder Node Resource Reference

This section lists all of the resources for a lightweight item node and resources inherited by the folder node. For resources specific to folder nodes, see Lightweight Folder Node Resource Reference on page 218.

Listed after the resource name are its data type, default value and a list of the procedures that may be used with the resource. **C** means it can be defined with a provided procedure (e.g. *XrtGearNodeCreateLW()*), **S** means there is a provided set procedure (e.g. *XrtGearNodeSetLabel()*), and **G** means there is a provided get procedure (e.g. *XrtGearNodeGetLabel()*).

Include File: `<Xm/XrtLWNode.h>`

Editable **Boolean** **True** **SG**

This resource controls whether or not users are allowed to edit node labels belonging to this node.

Label **XrtGearObject** **NULL** **CSG**

Specifies the current label being displayed.

Selected **Boolean** **False** **SG**

Specifies whether this node is selected (`True`).

Style **Widget** ***default node style*** **SG**

Specifies the NodeStyle object to be used as the default node style. If this resource is not set, the first element in the list specified by `XmNxrtGearNodeStyleList` is used as the default node style.

UserData **XtPointer** **NULL** **SG**

Provides a pointer to user-maintained data associated with a node. Note that the end-user is responsible for freeing any data placed here.

7.16.6 Procedures and Methods Reference

This section lists all of the procedures and methods available to a lightweight item node and those inherited by the folder node. For procedures and methods specific to folder nodes, see page 218.

Note, you can pass these procedures both XtObject nodes and lightweight nodes.

XrtGearIsNode()

Returns `True` if the given widget is an item node or a folder node.

```
Boolean
XrtGearIsNode(
     Widget              node
     )
```

XrtGearNodeCreateLW()

Creates a new lightweight node as a child of the given *parent*, assigning it the given *name* and *label*. The new node is retrieved, cast as a Widget for compatibility.

```
Widget
XrtGearNodeCreateLW(
     Widget              parent,
     String              name,
     XrtGearObject       label
     )
```

parent is a folder widget. The folder widget can be a lightweight folder object, a regular Outliner folder object, or the Outliner widget itself.

XrtGearNodeDestroy()

Destroys either an item or folder node.

```
void
XrtGearNodeDestroy(
     Widget              node,
     )
```

XrtGearNodeGetHeight()

Returns the indicated *node*'s height in pixels.

```
Dimension
XrtGearNodeGetHeight(
     Widget              node
)
```

XrtGearNodeGetEditable()

Returns the editability of the provided node.

```
Boolean
XrtGearNodeGetEditable(
    Widget              node
);
```

node can be either an XtObject or a Lightweight node.

XrtGearNodeGetLabel()

Returns the value of the `Label` resource of the indicated *node*.

```
XrtGearObject
XrtGearNodeGetLabel(
     Widget              node
     )
```

XrtGearNodeGetName()

Returns the value of the `Name` resource of the indicated *node*.

```
String
XrtGearNodeGetName(
     Widget              node
     )
```

XrtGearNodeGetNodeStyle()

Returns the value of the `Style` resource of the indicated *node*.

```
Widget
XrtGearNodeGetNodeStyle(
     Widget              node
     )
```

XrtGearNodeGetParent()

Returns the *node*'s parent. The parent can be a lightweight node, an XtObject node, or the Outliner widget.

```
Widget
XrtGearNodeGetParent(
     Widget              node
     )
```

XrtGearNodeGetSelected()

Returns the value of the `Selected` resource of the indicated *node*. Returns `True` if the node is selected.

```
Boolean
XrtGearNodeGetSelected(
     Widget              node
     )
```

XrtGearNodeGetUserData()

Returns a pointer to user-maintained data associated with the given *node*.

```
XtPointer
XrtGearNodeGetUserData(
     Widget              node
     )
```

XrtGearNodeGetWidth()

Returns the indicated *node*'s width in pixels.

```
Dimension
XrtGearNodeGetWidth(
     Widget              node
)
```

XrtGearNodeSetEditable()

Sets the editability of the provided node to the value of *editable*.

```
Boolean
XrtGearNodeSetSelected(
     Widget              node,
     Boolean             editable

     )
```

node can be either an XtObject or a Lightweight node.

XrtGearNodeSetLabel()

Sets the displayed *label* for the *node*.

```
Boolean
XrtGearNodeSetLabel(
     Widget              node,
     XrtGearObject       label
     )
```

XrtGearNodeSetNodeStyle()

Sets the node style.

```
Boolean
XrtGearNodeSetNodeStyle(
     Widget              node,
     Widget              style
     )
```

node is an item node. *style* is a node style object.

XrtGearNodeSetSelected()

Sets the *node*'s `Selected` resource to either `True` (selected) or `False` (not selected). Returns `True` if the attribute is successfully set.

```
Boolean
XrtGearNodeSetSelected(
     Widget              node,
     Boolean             selected
     )
```

node is an item node. *selected* is `Selected` resource's Boolean setting.

XrtGearNodeSetUserData()

Sets the *node*'s `UserData` attribute to a pointer to user-maintained data associated with the node. Note that the end-user is responsible for freeing any data placed here. Returns `True` if the attribute is set.

```
Boolean
XrtGearNodeSetUserData(
    Widget              node,
    XtPointer           user_data
    )
```

node is an item node. *user_data* a pointer to user-maintained data.

7.16.7 Lightweight Folder Node Resource Reference

This section lists resources specific to lightweight folder nodes. Additional lightweight folder node resources are listed under Resource Reference on page 151.

Listed after the resource name are its data type, default value and a list of the procedures that may be used with the resource. **C** means it can be defined with a provided procedure (e.g. *XrtGearFolderCreateLW()*), **S** means there is a provided set procedure (e.g. *XrtGearNodeChangeFolderState()*), and **G** means there is a provided get procedure (e.g. *XrtGearNodeGetFolderState()*).

Include File: `<Xm/XrtLWFolder.h>`

Children **XrtGearObject** **NULL** **G**

Specifies a pointer to an XrtList object that contains the item and folder nodes owned by the folder.

State **XrtGearFolderState** **NULL** **SG**

Specifies the current state of a folder node. Valid values are:

`XRTGEAR_FOLDERSTATE_CLOSED`	folder closed; no children visible
`XRTGEAR_FOLDERSTATE_OPEN_FOLDERS`	folder open; only folder children visible
`XRTGEAR_FOLDERSTATE_OPEN_ITEMS`	folder open; only item children visible
`XRTGEAR_FOLDERSTATE_OPEN_ALL`	folder open; all children visible

This value can be controlled programmatically or through user interaction. The Outliner widget's `XmNxrtGearFolderStateMask` resource controls which states are valid.

7.16.8 Procedures and Methods Reference

This section lists procedures and methods specific to lightweight folder nodes. Additional lightweight folder node procedures are listed on page 178.

Note, you can pass these procedures both XtObject nodes and lightweight nodes.

XrtGearFolderCreateLW()

Creates a lightweight folder node.

```
Widget
XrtGearFolderCreateLW(
     Widget             parent,
     String             name,
     XrtGearObject      label,
     XrtGearFolderState initial_state
     )
```

parent is the outliner widget. *name* is the created widget's name. *label* is the label text you wish to appear next to the folder. *initial_state* is the folder's `State` resource (for example, `XRTGEAR_FOLDERSTATE_CLOSED`).

XrtGearIsNodeFolder()

Returns `True` if *node* is a Folder or `False` if not.

```
Boolean
XrtGearIsNodeFolder(
     Widget             node
     )
```

node is a folder node.

XrtGearNodeChangeFolderState()

Sets the folder `state` resource. Note, this is the same procedure listed in the XRT/gear programmer's guide. It can now be used with lightweight folder nodes. Callbacks specified by `XmNxrtGearFolderStateCallback` are called as the state is changed.

```
void
XrtGearNodeChangeFolderState(
    Widget             folder,
    XrtGearFolderState state
    )
```

folder is the folder whose state is being changed. *state* is the new `State`; see the description of the `State` resource for the valid values of this parameter.

XrtGearNodeCreateLWTreeFromBuffer()

Creates a lightweight tree or subtree from a multi-line text string (assumed to be in data file format).

```
Widget
XrtGearNodeCreateLWTreeFromBuffer(
     Widget             parent,
     char               *buffer,
     Boolean            full_path,
     XrtGearFolderState initial_state,
     String             indent,
     String             separator,
     String             open_escape,
     String             close_escape
     )
```

parent is the parent of the newly-created tree. *buffer* is the multi-line text string; each line represents a node of the created tree.

The value returned by *XrtGearNodeCreateLWTreeFromBuffer()* is the parent or `Null` on error.

full_path indicates whether the complete tree hierarchy is provided for each node (*full_path* is `True`) or inferred from the indent levels (*full_path* is `False`). If *full_path* is `True`, the string specified by *indent* separates the tree hierarchy into its components. Creating a tree with *full_path* set to `True` can slow outliner. You should consider creating outliners with *full_path* set to `False`.

initial_state is the folder's `State` resource (for example, `XRTGEAR_FOLDERSTATE_CLOSED`).

indent is a string that specifies one indent level of the node on a particular line. If *indent* is `NULL`, "`\t`" (the tab character) is assumed.

separator is a string that breaks the text into discrete sections, which can be used to display a multicolumn outliner. The default is "`,`" (the comma character).

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is `NULL`, "`(`" is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is `NULL`, "`)`" is assumed.

XrtGearNodeCreateLWTreeFromStream()

Creates a tree from a data file or other input stream.

```
Widget
XrtGearNodeCreateLWTreeFromStream(
    Widget              parent,
    FILE                *stream,
    Boolean             full_path,
    XrtGearFolderState  initial_state,
    String              indent,
    String              separator,
    String              open_escape,
    String              close_escape
    )
```

parent is the parent of the newly-created tree. *stream* is the data file; each line of the file represents a node of the tree.

The value returned by *XrtGearNodeCreateLWTreeFromStream()* is the parent or `Null` on error.

stream is a pointer to a data file or other input stream.

full_path indicates whether the complete tree hierarchy is provided for each node (*full_path* is `True`) or inferred from the indent levels (*full_path* is `False`). If *full_path* is `True`, the string specified by *indent* separates the tree hierarchy into its components.

initial_state is the initial state of the folders in the tree.

indent is a string that specifies one indent level of the node on a particular line. If *indent* is `NULL`, "`\t`" (the tab character) is assumed.

separator is a string that breaks the text into discrete sections, which can be used to display a multicolumn outliner. The default is "`,`" (the comma character).

open_escape is a string that indicates the start of a sequence of instructions not belonging to the node text. If *open_escape* is `NULL`, "`(`" is assumed.

closed_escape is a string that indicates the end of a sequence of instructions not belonging to the node text. If *closed_escape* is `NULL`, "`)`" is assumed.

XrtGearNodeDestroy()

Destroys either an item or folder node. Note this is part of the item node API but is included here for completeness.

```
void
XrtGearNodeDestroy(
     Widget              node,
     )
```

XrtGearNodeGetChildren()

Returns a pointer to an XrtList object that contains the nodes to be drawn.

```
XrtGearObject
XrtGearNodeGetChildren(
     Widget              node
     )
```

XrtGearNodeGetFolderState()

Specifies the current state of a folder node.

```
XrtGearFolderState
XrtGearNodeGetFolderState(
     Widget              node
     )
```

node is a folder node. *XrtGearNodeGetFolderState()* returns:

`XRTGEAR_FOLDERSTATE_CLOSED`	folder closed; no children visible
`XRTGEAR_FOLDERSTATE_OPEN_FOLDERS`	folder open; only folder children visible
`XRTGEAR_FOLDERSTATE_OPEN_ITEMS`	folder open; only item children visible
`XRTGEAR_FOLDERSTATE_OPEN_ALL`	folder open; all folders and children visible

8

The Progress Widget

Sample Program ■ *Widget Synopsis*
Label and Range ■ *Timeouts and TimeOut Callbacks*
Label Display ■ *Bar Display*
Resource Reference ■ *Procedures and Methods Reference*

8.1 Sample Program

The Progress widget enables you to update the end-user on the status of a task. The following program is a simple application that uses a Progress widget. The code that creates and modifies the Progress widget itself is shown in boldface.

```
#include <stdio.h>
#include <stdlib.h>
#include <Xm/XrtGear.h>
#include <Xm/XrtProgress.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>

Widget progress;
Display *display;

static void
buttonCB(Widget w, XtPointer client_data, XmAnyCallbackStruct *cb)
{
    int i = 0;

    for (i=1; i<=100; i++) {
        XtVaSetValues(progress,
            XmNvalue, i,
            NULL);
        XSync(display, True);
        usleep(100000);
    }
    exit(0);
}
```

```
void
main(int argc, char **argv)
{
    Widget toplevel, container, buttonArea, button;
    XtAppContext app;

    toplevel = XtVaAppInitialize(&app, "Progress1", NULL, 0,
                                 &argc, argv, NULL, NULL);
    container = XtVaCreateManagedWidget( "MyContainer",
        xmFormWidgetClass,      toplevel,
        XmNorientation,         XmVERTICAL,
        NULL);
    buttonArea = XtVaCreateManagedWidget( "MyButtonArea",
        xmRowColumnWidgetClass, container,
        XmNtopAttachment,       XmATTACH_FORM,
        XmNleftAttachment,      XmATTACH_FORM,
        XmNrightAttachment,     XmATTACH_FORM,
        NULL);
    button = XtVaCreateManagedWidget("Press to start",
        xmPushButtonWidgetClass,buttonArea,
        NULL);
    XtAddCallback(button, XmNactivateCallback,
        (XtCallbackProc) buttonCB, NULL);
    progress = XtVaCreateWidget("prog",
        xmXrtProgressWidgetClass, container,
        XmNtopAttachment,       XmATTACH_WIDGET,
        XmNtopWidget,           buttonArea,
        XmNleftAttachment,      XmATTACH_FORM,
        XmNrightAttachment,     XmATTACH_FORM,
        XmNbottomAttachment,    XmATTACH_FORM,
        NULL);
    display = XtDisplay(progress);
    XtManageChild(progress);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}
```

When this program is run, the following appears:

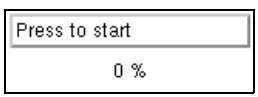


Figure 72 A sample Progress widget in its initial state

When the end-user presses the `Press to start` button, the value of the Progress widget's `XmNvalue` resource is updated, and the widget redisplayed every tenth of a second. The `XmNvalue` resource contains the current value of the Progress widget, which is a number between 0 and 100 representing a percentage. This current percentage is displayed in two ways: as a numeric value and as a horizontal bar, as shown in Figure 73.

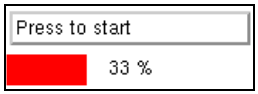


Figure 73 A Progress widget displaying a value

In more complicated real-world applications, a pointer to the Progress widget can be passed to another application window, which changes the value of `XmNvalue` when appropriate. This enables the application to use the Progress widget to monitor the completion of a task.

The following sections describe how to change the default behavior of the Progress widget, describe how to update the Progress widget using timeouts and TimeOut callbacks, and provide other examples of how to use the widget.

8.2 Widget Synopsis

Include Files:	*<Xm/XrtProgress.h>*
Class Name:	XmXrtProgress
Class Hierarchy:	Core → XmPrimitive → XmXrtProgress
Class Pointer:	xmXrtProgressWidgetClass

8.3 Label and Range

The value displayed by the Progress widget is controlled by three resources:

- The value itself is stored in the `XmNvalue` resource.
- The maximum value allowed is specified by the `XmNmaximum` resource.
- The minimum value allowed is specified by the `XmNminimum` resource.

The `XmNvalue` resource stores the Progress widget's current value, and is of type `int`. This value is formatted by the widget before being displayed: for instance, in the previous example the value 43 is formatted to become the string `43%`. The resulting formatted string, called the *label*, is stored in the `XmNxrtGearLabel` resource, which can be read but not set. (For more information on value formatting, see section 8.5 on page 228.)

The `XmNminimum` resource specifies the smallest integer that can be stored in `XmNvalue`. By default, this value is 0. When the Progress widget is realized, the value of `XmNminimum` becomes the initial value of `XmNvalue`. If `XmNvalue` is set to a value smaller than `XmNminimum`, the value of `XmNminimum` is used.

The `XmNmaximum` resource specifies the largest integer that can be stored in `XmNvalue`. By default, this value is 100. When an application sets `XmNvalue` to the value of `XmNmaximum`, the task whose progress is measured by the Progress widget is assumed

to be complete. At this point, the callbacks specified by XmNxrtGearProgressCallback are called; the supplied callback reason is XRTGEAR_REASON_COMPLETE. (Progress callbacks are discussed in more detail in section 8.4 on page 226.) If XmNvalue is set to a value greater than XmNmaximum, the value of XmNmaximum is used.

For example, the following code modifies the Progress widget to serve as a monitor for an eight-step process:

```
XtVaSetValues(progress,
    XmNminimum,     1,
    XmNmaximum,     8,
    NULL);
```

XmNvalue can now contain any integer between 1 and 8 (inclusive).

Normally, the application slowly increases the value of XmNvalue from the minimum value to the maximum value.

The Progress Percentage

The default minimum and maximum of 0 and 100 make it easy to use the Progress widget to indicate what percentage of a task has been performed, since the value of XmNvalue is equivalent to the percentage. However, if XmNminimum or XmNmaximum is changed to something other than the default value, the value of XmNvalue may not be the same as the task percentage.

To make it easy to use the Progress widget to chart task percentage when the minimum and maximum are not 0 and 100, the task percentage is stored as a double-precision floating-point number in the XmNxrtGearValuePercent resource. XmNxrtGearValuePercent calculates the percentage using the following formula:

```
percentage = XmNvalue / (XmNmaximum - XmNminimum)
```

The value of XmNxrtGearPercent is recalculated whenever XmNvalue, XmNmaximum or XmNminimum is changed.

8.4 Timeouts and TimeOut Callbacks

In the example in section 8.1 on page 223, the value of the Progress widget is updated by changing the value of its XmNvalue resource. You can also change the value of the Progress widget using the widget's timeout mechanism, by doing the following:

- Defining a callback routine that changes the value, adding it to the list defined by the XmNxrtGearProgressCallback resource, and then
- Defining a timeout interval by setting the XmNxrtGearProgressTimeOut resource.

XmNxrtGearProgressTimeOut specifies the time, in milliseconds, between timeouts; when a timeout occurs, the callback routine is called. If this resource is set to 0 (its default value), no timeouts occur.

Each callback on the XmNxrtGearProgressCallback list is called at three points:

- When a timeout process is started (when XmNxrtGearProgressTimeOut is set to a non-zero value): at this point, the *reason* member of the callback structure is set to XRTGEAR_REASON_PROGRESS_BEGIN.
- When a timeout occurs: *reason* is set to XRTGEAR_REASON_PROGRESS. At this time, the value of the widget can be changed.
- When the widget reaches its maximum value (XmNvalue is equal to XmNmaximum): *reason* is set to XRTGEAR_REASON_PROGRESS_COMPLETE. This call happens even if XmNxrtGearProgressTimeOut has the value 0.

When a TimeOut callback is called, it is passed the following structure:

```
typedef struct {
    XrtGearReason       reason;
    XEvent              *event;
    int                 minimum;
    int                 maximum;
    int                 value_percent;
    XmString            label;
    int                 value;
    Boolean             continue_progression;
} XrtGearProgressCallbackStruct;
```

These fields are all read-only, except *label*, *value* and *continue_progression*, which can be modified when reason is XRTGEAR_REASON_PROGRESS. Setting *continue_progression* to False stops the progression currently in progress.

The following is example code that uses TimeOut callbacks to control the value of the Progress widget.

```
static void
progressCB(Widget w, XtPointer client_data, XtPointer call_data)
{
    XrtGearProgressCallbackStruct *cbs =
        (XrtGearProgressCallbackStruct *) call_data;

    if (cbs->reason == XRTGEAR_REASON_PROGRESS) {
        cbs->value += 1;
    } else if (cbs->reason == XRTGEAR_REASON_PROGRESS_COMPLETE) {
        exit(0);
    }
}

static void
start_timerCB(Widget w, XtPointer client_data,
    XmAnyCallbackStruct *cb)
{
    if (progress != NULL) {
        XtVaSetValues(progress,
            XmNxrtGearProgressTimeOut, 100,
        NULL);
    }
    XtAddCallback(progress, XmNxrtGearProgressCallback,
        progressCB, NULL);
}
```

```
static void
stop_timerCB(Widget w, XtPointer client_data,
    XmAnyCallbackStruct *cb)
{
    if (progress != NULL) {
        XtVaSetValues(progress,
            XmNxrtGearProgressTimeOut, 0,
        NULL);
    }
}

...

    progress = XtVaCreateWidget("prog",
        xmXrtProgressWidgetClass, container,
        XmNtopAttachment,       XmATTACH_WIDGET,
        XmNtopWidget,           buttonArea,
        XmNleftAttachment,      XmATTACH_FORM,
        XmNrightAttachment,     XmATTACH_FORM,
        XmNbottomAttachment,    XmATTACH_FORM,
        NULL);
```

The percentage displayed in the Progress widget is incremented every tenth of a second until it reaches 100, at which point the widget exits. The value is incremented by modifying the *value* member of the TimeOut callback structure from within the `progressCB` routine.

Destroying the Widget

Note that you cannot destroy the Progress widget from within any callback. The easiest way to create a Progress widget that destroys itself once the progression has completed is to use the *XmCreateXrtProgressShellSimple()* procedure:

```
Arg args[3];
int n = 0;

XtSetArg(args[n], XmNminimum, 1); n++;
XtSetArg(args[n], XmNmaximum, 10); n++;
progress = XmCreateXrtProgressShellSimple(parent,
    "myshelltitle", args, n);
```

This creates a progress widget in a window named “myshelltitle”, which is automatically destroyed when the progression has completed (when `XmNvalue` is equivalent to `XmNmaximum`).

You can also destroy a Progress widget from within a timeout procedure created by calling *XtAppAddTimeOut()*, or from a work procedure created by calling *XtAppAddWorkProc()*.

8.5 Label Display

The Progress widget provides several resources that control how the Progress widget’s label (the current value formatted as a string) is displayed. Changing these resources enables you to specify:

- Whether the label is to be displayed at all

- The print format to use when displaying the label
- Customized text to appear at particular points in the progress (using LabelFormat callbacks)
- The font, alignment and color of the label
- Whether the widget is to change its size so that the string or pixmap fits

Suppressing Label Display

By default, the label is displayed whenever the Progress widget is realized or updated. To turn off label display, set `XmNxrtGearLabelShow` to `False`. This tells the Progress widget to display only the bar.

Setting `XmNxrtGearLabelShow` to `False` is useful if you want to display the label in some place other than the Progress widget. (See the description of LabelFormat callbacks later in this section for details on how to do this.)

Label Formatting

By default, when the current value of the Progress widget is formatted into a label for display purposes, a percent sign `%` is appended to the label. You can specify the label format string to be used when formatting the label by setting the `XmNxrtGearLabelFormat` resource.

This resource expects a string as its value. This string can contain one or more of the following special format sequences:

`%C`	estimated time to completion (displayed as *hours*:*minutes*:*seconds*, with *hours* and *minutes* omitted when not needed)
`%E`	estimated total time
`%M`	the maximum value of the label (the value of `XmNmaximum`)
`%N`	the minimum value (the value of `XmNminimum`)
`%P`	the percentage (the value of `XmNxrtGearValuePercent` multiplied by 100)
`%T`	the elapsed time
`%V`	the current value (the value of `XmNvalue`)
`%%`	a single percent character

The following are examples of some possible format strings:

```
"processing part %V of %M"
"%C remaining"
"time: %T of %E"
```

The default value for `XmNxrtGearLabelFormat` is `"%P %%"` (the current percentage followed by a space and a percent sign).

The elapsed time, total time and estimated time values are also obtainable using convenience methods. See section 8.8 on page 238 for more details on these methods. (Note that the total time and estimated time values become more accurate as the progression continues.)

Label Format Callbacks

The Progress widget enables you to modify the label to be displayed, or retrieve the current formatted label and store it in another widget. These tasks can be accomplished by defining callbacks using the `XmNxrtGearLabelFormatCallback` resource.

Each callback in the list is passed an `XrtGearProgressCallbackStruct` structure:

```
typedef struct {
    XrtGearReason       reason;
    XEvent              *event;
    int                 minimum;
    int                 maximum;
    int                 value_percent;
    XmString            label;
    int                 value;
    Boolean             continue_progression;
} XrtGearProgressCallbackStruct;
```

Each callback is called twice per label change: once before the new label is created, and once after it is created. The *reason* element of the callback structure is set to `XRTGEAR_REASON_FORMAT_LABEL_BEGIN` for the first call, and `XRTGEAR_REASON_FORMAT_LABEL_END` for the second. When *reason* is `XRTGEAR_REASON_FORMAT_LABEL_BEGIN`, you can modify the label to be displayed by changing the *label* element of the callback structure.

Storing the Formatted Label in Another Widget

When a LabelFormat callback is called with *reason* set to `XRTGEAR_REASON_FORMAT_LABEL_END`, it can be used to retrieve the current formatted label and store it in another widget. For example, the following code retrieves the current label and stores it as the label string in an XmLabel widget:

```
void
labelCB(Widget w, XtPointer client_data,
    XrtGearProgressCallbackStruct *call_data)
{
    if (call_data->reason == XRTGEAR_REASON_FORMAT_LABEL_END) {
        XtVaSetValues(label,
            XmNlabelString, call_data->label,
            NULL);
    }
}
```

This code displays the label in the specified XmLabel widget.

Label Font

To specify the font for the label text, set the `XmNfontList` resource, as shown in the following example:

```
XtVaSetValues(progress,
    XtVaTypedArg,   XmNfontList,
    XmRString, "-*-times-medium-*-*--12-*-iso8859-*",
        strlen("-*-times-medium-*-*--12-*-iso8859-*") + 1,
    NULL);
```

Label Alignment

By default, the label text appears in the center of the Progress widget's display area. To change the alignment of label text, set the `XmNxrtGearLabelAlignment` resource, as shown in Figure 74.

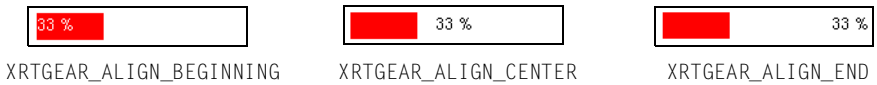


Figure 74 Progress widget label alignment

The default value is `XRTGEAR_ALIGN_CENTER`.

Label Color

The color of the label text depends on whether the text is contained in a bar. Any portion of the text that is contained in the bar is drawn in the widget background color (specified by `XmNbackground`). All other label text is drawn in the widget's foreground color (specified by `XmNforeground`).

Recomputing Size

The `XmNrecomputeSize` resource controls whether the widget changes its size to ensure that the label string exactly fits. By default, this resource is set to `True`, and size recomputing is performed.

8.6 Bar Display

The Progress widget enables you to control the following aspects of bar display:

- The type of bar displayed
- The margin height and width
- The number of bars to be displayed and the space between bars (if the bar consists of discrete components)
- The icon to use (if the bar display type uses an icon)
- The bar color

Bar Type

By default, the bar displayed by the Progress widget is a continuous horizontal bar. To change the bar type, set the `XmNxrtGearProgressType` widget. The valid values for this resource, and their effects, are shown in Figure 75.

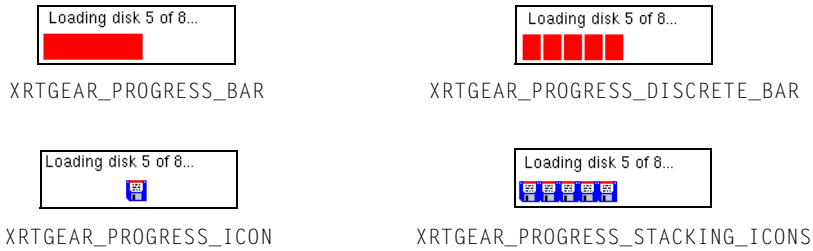


Figure 75 Bar types

The default value is `XRTGEAR_PROGRESS_BAR`.

If the bar type is affected by spacing restrictions, the following rules apply:

- If `XmNxrtGearProgressType` is `XRTGEAR_PROGRESS_DISCRETE_BAR` and the discrete bars cannot be displayed, `XRTGEAR_PROGRESS_BAR` is used.
- If `XmNxrtGearProgressType` is `XRTGEAR_PROGRESS_STACKING_ICONS` and the stacking icons cannot be displayed, `XRTGEAR_PROGRESS_ICON` is used.
- If `XmNxrtGearProgressType` is `XRTGEAR_PROGRESS_ICON` and the icon cannot be displayed, `XRTGEAR_PROGRESS_BAR` is used.

Margin Height and Width

The `XmNmarginHeight` and `XmNmarginWidth` resources control the amount of space between the border of the widget and the displayed bar. By default, both margins are set to two pixels. The effects of changing these resources are shown in Figure 76.

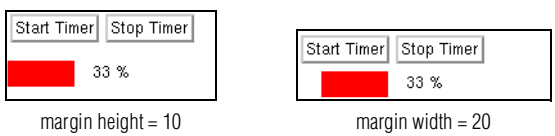


Figure 76 Margin height and width

Bar Count

When `XmNxrtGearProgressType` is set to `XRTGEAR_PROGRESS_DISCRETE_BAR`, the maximum number of discrete bars to be displayed is specified by the `XmNxrtGearBarCount` resource. Figure 77 shows the effect of this resource.

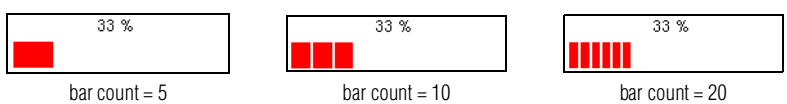


Figure 77 Discrete bar count

By default, XmNxrtGearBarCount is 10. This means that a maximum of 10 bars or icons will be displayed. (The number of bars/icons actually displayed is proportional to the percentage of the task that has been completed.)

Bar Spacing

When XrtGearProgressType is either XRTGEAR_PROGRESS_DISCRETE_BAR or XRTGEAR_PROGRESS_STACKING_ICONS, a space of two pixels is left between individual bars or icons by default. To change this spacing, set the XmNxrtGearBarSpacing resource, as shown in Figure 78.

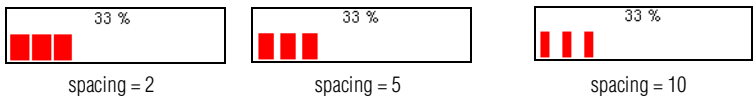


Figure 78 Discrete bar spacing

Note that a solid bar is used if the bar count or bar spacing is set to a value that makes discrete bars impossible to see.

Bar Icon

To specify the icon to be displayed when the bar type is set to XRTGEAR_PROGRESS_ICON or XRTGEAR_PROGRESS_STACKING_ICONS, set the XmNxrtGearIcon resource. This resource expects a pointer to a value of type XrtGearIcon.

For a description of methods that create and manipulate icons, see section 13.4 on page 300.

Bar Color

To specify the color in which the bar is drawn, set the XmNxrtGearBarColor resource. This resource is of type Pixel, and is set in the same way as standard Motif color resources such as XmNbackground.

8.7 Resource Reference

This section lists all of the resources for the Progress widget, plus some of the more commonly used inherited resources. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

XmNxrtGearBarColor **Pixel** **Red** **CSG**

Specifies the color of the bar(s) displayed by the Progress widget. Applies when XmNxrtGear-ProgressType is XRTGEAR_PROGRESS_BAR or XRTGEAR_PROGRESS_DISCRETE_BAR.

XmNxrtGearBarCount **int** **10** **CSG**

Specifies the number of discrete bars eventually displayed by the Progress widget. Applies when XmNxrtGearProgressType is XRTGEAR_PROGRESS_DISCRETE_BAR.

XmNxrtGearBarSpacing **Dimension** **2** **CSG**

Specifies the space in pixels between each unit. Applies when XmNxrtGearProgressType is either XRTGEAR_PROGRESS_DISCRETE_BAR or XRTGEAR_PROGRESS_STACKING_ICONS.

If this resource is set to a value too large to display all the bars or icons, the spacing is automatically reduced to ensure that all the bars or icons fit in the widget.

XmNxrtGearDoubleBuffer **Boolean** **True** **CSG**

Specifies whether or not the widget should be double-buffered for better performance. Double-buffering requires the use of more Server memory.

XmNfontList **XmFontList** ***dynamic*** **CSG**

Specifies the font list used for the label text.

XmNxrtGearIcon **XrtGearIcon** **NULL** **CSG**

Specifies the image used when XmNxrtGearProgressType is XRTGEAR_PROGRESS_ICON.

XmNxrtGearLabel **XmString** **NULL** **G**

Specifies the current value label being displayed.

XmNxrtGearLabelAlignment **XrtGearAlignment** **XRTGEAR_ALIGN_CENTER** **CSG**

Specifies the location of the label in the bar. Possible values are XRTGEAR_ALIGN_BEGINNING, XRTGEAR_ALIGN_CENTER, and XRTGEAR_ALIGN_END.

XmNxrtGearLabelFormat **String** **"%P %%"** **CSG**

Specifies the format of the value when it is displayed in the progress area.

The following special character sequences are defined:

%C	estimated time to completion (displayed as *hours*:*minutes*:*seconds*, with *hours* and *minutes* omitted when not needed)
%E	estimated total time
%M	the maximum value of the label (the value of XmNmaximum)
%N	the minimum value (the value of XmNminimum)
%P	the percentage (the value of XmNxrtGearValuePercent multiplied by 100)
%T	the elapsed time
%V	the current value (the value of XmNvalue)
%%	a single percent character

XmNxrtGearLabelFormatCallback **XtCallbackList** **NULL** **G**

Specifies the callbacks to call when creating a new format value. These callbacks are called whenever the XmNvalue resource changes. Each routine is passed a pointer to an XrtGearProgressCallbackStruct. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget;
    XtPointer   client_data;
    XtPointer   call_data /* points to an
                             XrtGearProgressCallbackStruct */
)
```

where XrtGearProgressCallbackStruct is defined as

```
typedef struct {
    XrtGearReason       reason;
    XEvent              *event;
    int                 minimum;
    int                 maximum;
    int                 value_percent;
    XmString            label;
    int                 value;
    Boolean             continue_progression;
} XrtGearProgressCallbackStruct;
```

reason is either XRTGEAR_REASON_FORMAT_LABEL_BEGIN (new label about to be created) or XRTGEAR_REASON_FORMAT_LABEL_END (after label created). When *reason* is XRTGEAR_REASON_FORMAT_LABEL_BEGIN, you can specify a label by changing the *label* member of the callback structure. *event* is the XEvent that triggered the callback, or NULL if the callback is called programmatically. (The other fields are used by Progress callbacks.)

XmNxrtGearLabelShow **Boolean** **True** **CSG**

Specifies whether to display the label (True if displayed). The value of the label can be obtained by supplying a callback procedure to the XmNxrtGearLabelFormatCallback resource.

XmNmarginHeight **Dimension** **2** **CSG**

Specifies the space between the top and bottom edges of the border and the bar/icons.

XmNmarginWidth **Dimension** **2** **CSG**

Specifies the space between the left and right edges of the border and the bar/icons.

XmNmaximum **int** **100** **CSG**

Specifies the maximum allowed value of XmNvalue. If the value of XmNvalue becomes equal to the value of XmNmaximum, the callbacks specified by XmNxrtGearProgressCallback are called with the reason XRTGEAR_REASON_COMPLETE.

XmNminimum **int** **0** **CSG**

Specifies the minimum allowed value of XmNvalue.

XmNxrtGearProgressCallback **XtCallbackList** **NULL** **CSG**

Specifies callbacks to be invoked whenever the progress state changes. Each routine is passed a pointer to an `XrtGearProgressCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget;
    XtPointer   client_data;
    XtPointer   call_data /* points to an
                             XrtGearProgressCallbackStruct */
)
```

where `XrtGearProgressCallbackStruct` is defined as

```
typedef struct {
    XrtGearReason       reason;                 /* read-only */
    XEvent              *event;                 /* read-only */
    int                 minimum;                /* read-only */
    int                 maximum;                /* read-only */
    int                 value_percent;          /* read-only */
    XmString            label;
    int                 value;
    Boolean             continue_progression;
} XrtGearProgressCallbackStruct;
```

These callbacks are called at three points. First, they are called when the widget begins progressing. In this case, *reason* is set to `XRTGEAR_REASON_PROGRESS_BEGIN`.

The callbacks are also called every time a timeout occurs. (The time interval between timeouts is specified by the `XmNxrtGearProgressTimeOut` resource.) In this case, *reason* is `XRTGEAR_REASON_PROGRESS`, and *value* is writeable: the programmer may change *value* to update the current status or progress of the widget.

The callbacks are also called when `XmNvalue` is set to the value of `XmNmaximum`. In this case, *reason* is `XRTGEAR_REASON_PROGRESS_COMPLETE`.

event is the XEvent that triggered the callback, or `NULL` if the callback is called programmatically; *minimum* and *maximum* are the minimum and maximum values for this progression; *value_percent* is the percentage of the progression currently completed.

To stop a progression, set *continue_progression* to `False` when *reason* is `XRTGEAR_REASON_PROGRESS`.

XmNxrtGearProgressTimeOut **unsigned long** **0** **CSG**

Specifies the amount of time between timeouts, measured in milliseconds. When a timeout occurs, the timeout callbacks defined by `XmNxrtGearProgressCallback` are called.

If this resource is set to 0 (its default value), no timeouts occur.

XmNxrtGearProgressType **XrtGearProgressType** **XRTGEAR_PROGRESS_BAR** **CSG**

Specifies the display type of the progress widget. Valid values are:

XRTGEAR_PROGRESS_BAR	single horizontal bar
XRTGEAR_PROGRESS_DISCRETE_BAR	discrete horizontal bars
XRTGEAR_PROGRESS_ICON	single icon that moves to indicate progress
XRTGEAR_PROGRESS_STACKING_ICONS	discrete icons

If the bar type is affected by spacing restrictions, the following rules apply:

- If XmNxrtGearProgressType is XRTGEAR_PROGRESS_DISCRETE_BAR and the discrete bars cannot be displayed, XRTGEAR_PROGRESS_BAR is used.
- If XmNxrtGearProgressType is XRTGEAR_PROGRESS_STACKING_ICONS and the stacking icons cannot be displayed, XRTGEAR_PROGRESS_ICON is used.
- If XmNxrtGearProgressType is XRTGEAR_PROGRESS_ICON and the icon cannot be displayed, XRTGEAR_PROGRESS_BAR is used.

XmNrecomputeSize **Boolean** **True** **CSG**

Specifies whether the widget is to change its size so that the string or pixmap fits.

XmNvalue **int** **0** **CSG**

Specifies the current value. This must be equal to or greater than the value of XmNminimum and less than or equal to the value of XmNmaximum. If the value is outside this range, it will be set to the closest value inside the range, and a warning message will be output.

XmNxrtGearValuePercent **double** **0** **G**

Specifies the widget progress, measured as a percentage between 0 and 1. This percentage is calculated as follows:

```
percentage = XmNvalue / (XmNmaximum - XmNminimum)
```

If the value of XmNvalue, XmNmaximum or XmNminimum changes, this percentage is recalculated.

8.7.1 Resources Inherited by the Progress Widget

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	Primitive
XmNbottomShadowPixmap	Primitive
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	Primitive
XmNheight	Core
XmNhelpCallback	Primitive
XmNhighlightColor	Primitive

Resource	Inherited From
XmNhighlightOnEnter	Primitive
XmNhighlightPixmap	Primitive
XmNinitialResourcesPersistent	Core
XmNmappedWhenManaged	Core
XmNnavigationType	Primitive
XmNscreen	Core
XmNsensitive	Core
XmNtopShadowColor	Primitive
XmNtopShadowPixmap	Primitive
XmNtranslations	Core
XmNtraversalOn	Primitive
XmNuserData	Primitive
XmNwidth	Core
XmNx	Core
XmNy	Core

8.8 Procedures and Methods Reference

This section lists the Progress procedures and methods in alphabetical order.

XmCreateXrtProgress()
Creates a Progress widget.

```
Widget
XmCreateXrtProgress(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of *argcount* arguments.

XmCreateXrtProgressShellSimple()

Creates a shell containing a Progress widget, and destroys it when the progression has completed.

```
Widget
XmCreateXrtProgressShellSimple(
    Widget  parent,
    char    *title,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent; *title* is the title for the shell; *arglist* is a list of *argcount* arguments.

XmIsXrtProgress()

Returns `True` if a widget is a Progress widget.

```
Boolean
XmIsXrtProgress(
    Widget  widget
    )
```

XrtGearProgressCalcTimeElapsed()

Calculates the time elapsed since `XmNvalue` was last set to the value of `XmNminimum`.

```
time_t
XrtGearProgressCalcTimeElapsed(
    Widget  progress
    )
```

progress is a Progress widget.

XrtGearProgressCalcTimeToCompletion()

Calculates an estimate of the time required to complete the current progression.

```
time_t
XrtGearProgressCalcTimeToCompletion(
    Widget  progress
    )
```

progress is a Progress widget. The estimate improves as the progression continues.

XrtGearProgressCalcTimeTotal()

Calculates an estimate of the total time taken by the current progression.

```
time_t
XrtGearProgressCalcTimeTotal(
    Widget  progress
    )
```

progress is a Progress widget. The estimate improves as the progression continues.

9

The Combo Box Widget

Sample Program ■ *Widget Synopsis*
Menu Processing ■ *Controlling Display Attributes*
User Interaction ■ *Auto Matching Completion*
Resource Reference ■ *Procedures and Methods Reference*
Translations and Actions

9.1 Sample Program

The Combo Box widget provides a convenient way to enable end-users to choose from a list of alternatives. The following is a simple program that creates a Combo Box widget.

```
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Text.h>
#include <Xm/XrtGear.h>
#include <Xm/XrtGearCombo.h>

int
main (int argc, char *argv[])
{
        XtAppContext    app_context;
        Widget          toplevel, combobox;
        static String picklist[] = {"John Jones","Harry Smith",
            "Jean Jones","Mary Martin","Harry Hayes",NULL};

        toplevel = XtVaAppInitialize(&app_context,
            "Combo", NULL, 0,
            &argc, argv, NULL, NULL, 0);

        combobox = XtVaCreateManagedWidget("combobox",
            xmXrtGearComboWidgetClass,      toplevel,
            XmNxrtGearPickList,             picklist,
            NULL);

        XtRealizeWidget(toplevel);
        XtAppMainLoop(app_context);
}
```

When this program is compiled and run, the following appears on the screen:



Figure 79 The Combo Box widget created by this program

When the Combo Box widget is created, two child widgets are also created: a Text widget, called the *text field*, which contains the currently selected item, and an arrow widget that the end-user clicks on to change the current item. When the Combo Box first appears, the text field is empty, since the end-user has not yet selected an item.

In this program, the `XmNxrtGearPickList` resource defines a list of legal values for the Combo Box. When the user clicks on the arrow widget, a popup menu (of class XmList) appears, which contains the list of legal values defined by `XmNxrtGearPickList`. This list, called a *pick list*, is displayed in Figure 80.

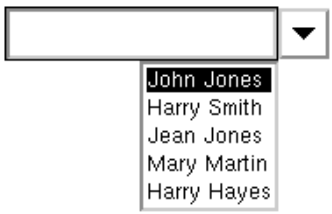


Figure 80 A popup menu contained in a Combo Box widget

The end-user can select an item from the pick list by clicking on it. Once the end-user has selected an item, the popup menu disappears and the text field contains the selected value. The end-user can make the menu disappear without changing the text field by clicking on the arrow key again.

9.2 Widget Synopsis

Include Files:	*<Xm/XrtGearCombo.h>*
Class Name:	XmXrtGearCombo
Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtManager → XmXrtComboBox
Class Pointer:	xmXrtGearComboWidgetClass

9.3 Menu Processing

The Combo Box widget enables you to control the following pick list attributes:

- The contents of the pick list
- The selected element of the pick list
- The widget ID of the pick list
- The widget ID of the text field containing the selected element

In addition, you can supply callbacks to be invoked when a menu item is selected.

9.3.1 Specifying the Pick List

To specify the pick list to appear in the popup menu, create an array of Strings and set the `XmNxrtGearPickList` resource to point to this array. This list can be changed after the resource has been created, which enables you to change the popup menu to respond to external events or end-user actions.

The Pick List Index

To select an item from the popup menu, set the `XmNxrtGearPickListIndex` resource. For example, to indicate that the first item in the popup menu is to be selected, set this resource to 0. Similarly, set this resource to 1 to indicate that the second item in the popup menu is to be selected, and so on. When the end-user selects an item from the popup menu, `XmNxrtGearPickListIndex` changes to reflect this selection.

If no item is displayed in the list, `XmNxrtGearPickListIndex` is set to `XRTGEAR_NOVALUE`.

Modifying the Pick List

To obtain the widget ID of the pick list, get the value of the `XmNxrtGearMenuList` resource. This resource is read-only; it cannot be set.

The pick list is an XmList widget; once you have its widget ID, you can set any resources not controlled by the parent Combo Box widget. The following resources are controlled by the parent widget and cannot be set programmatically: `XmNborderWidth`, `XmNheight`, `XmNitemCount`, `XmNitems`, `XmNselectionPolicy`, `XmNvisibleItemCount`, and `XmNwidth`.

The following example specifeis the font in which the pick list items are to be displayed by setting the pick list's `XmNfontList` resource:

```
static String fontlist =
"-adobe-helvetica-medium-r-normal--12-120-75-75-p-67-iso8859-1,\
-adobe-helvetica-bold-o-normal--14-140-75-75-p-82-iso8859\
-1=highlight";
XtVaGetValues(combobox,
    XmNxrtGearMenuList, &menulist,
    NULL);
XtVaSetValues(menulist,
    XtVaTypedArg, XmNfontList, XmRString,
    fontlist, strlen(fontlist)+1,
    NULL);
```

Note that if XmNxrtGearSearchPolicy is set to XRTGEAR_SEARCH_MULTICHAR, the highlight font is used to display keys entered by the end-user when searching in a pick list, as shown in Figure 81. (For more information on XmNxrtGearSearchPolicy, see section 9.5 on page 248.)

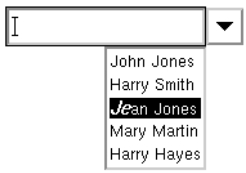


Figure 81 Using a highlight font in a pick list

For more information on Combo Box user interaction, see section 9.5 on page 248.

Modifying the Text Field

When a Combo Box widget is created, a child Text widget is also automatically created; this Text widget contains the currently selected item.

The following Text resources can be changed or retrieved by passing the combo box widget ID to *XtVaSetValues()* or *XtVaGetValues()*:

XmNblinkRate	XmNcolumns
XmNcursorPosition	XmNcursorPositionVisible
XmNeditable	XmNfontList
XmNmarginHeight	XmNmarginWidth
XmNmaxLength	XmNpendingDelete
XmNresizeWidth	XmNselectThreshold
XmNselectionArray	XmNselectionArrayCount
XmNvalue	XmNvalueWcs
XmNverifyBell	

To set other text resources or add callbacks, you must retrieve the widget ID of the text field. To do this, get the value of the XmNxrtGearTextField resource. This resource is read-only; it cannot be set.

9.3.2 Menu Callbacks

You can provide a list of callbacks to be invoked whenever the popup menu appears or disappears. This list is stored in the `XmNxrtGearMenuCallback` resource. Each callback in this list is passed the following structure:

```
typedef struct {
    int         reason;         /* read-only */
    XEvent      *event;         /* read-only */
    int         position;       /* read-only */
    String      value;
    Boolean     doit;           /* initially True */
} XrtGearMenuCallbackStruct;
```

The *reason* element indicates at what point the callback is being called. If the menu is about to appear, *reason* is `XRTGEAR_REASON_MENU_BEGIN`. If the menu is about to disappear, *reason* is `XRTGEAR_REASON_MENU_END`. The *position* element indicates the currently-selected item, and *value* is the string that is displayed (or about to be displayed) in the text field.

You can use menu callbacks to either suppress the display of the popup menu or disallow an end-user's menu item selection. To suppress popup display, set *doit* to `False` when *reason* is `XRTGEAR_REASON_MENU_BEGIN`. To ignore an end-user's menu item selection, set *doit* to `False` when reason is `XRTGEAR_REASON_MENU_END`.

9.4 Controlling Display Attributes

The Combo Box widget enables you to control the following elements of its appearance:

- The type of arrow to be displayed
- The widget class of the arrow at creation time
- The alignment of the popup menu to the text field
- The spacing between the text field and the arrow widget
- The number of items visible in the drop-down list

9.4.1 Arrow Display

To specify the type of arrow to be displayed, see the `XmNxrtGearArrowType` resource. This resource can only be set at the time the widget is created. Valid values are shown in Figure 82.



Figure 82 Combo Box arrow types

The default value is `XRTGEAR_ARROW_WEDGE`.

The position of the arrow button relative to the text field is determined by the value of XmNstringDirection.

In addition to the above arrow types, you can also set XmNxrtGearArrowType to XRTGEAR_ARROW_CUSTOM. This indicates that you are supplying your own arrow type. The following code is a sample program that builds a custom arrow type:

```
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/ArrowB.h>
#include <Xm/XrtGearCombo.h>

static void exposeArrow(Widget w, XtPointer client_data,
    XtPointer call_data);

int
main(int argc, char *argv[])
{
    XtAppContext    app_context;
    Widget          combo, toplevel, arrow = NULL;
    static String picklist[] =
        { "Joe Blow", "Harry Wong", "Sally Ann", NULL };
    Widget          incr = NULL;

    toplevel = XtVaAppInitialize(&app_context, "Combo", NULL, 0,
        &argc, argv, NULL, NULL);
    combo = XtVaCreateManagedWidget("combo",
        xmXrtGearComboWidgetClass,toplevel,
        XmNxrtGearPickList,         picklist,
        XmNxrtGearArrowType,        XRTGEAR_ARROW_CUSTOM,
        NULL);
    XtVaGetValues(combo, XmNxrtGearArrow, &arrow, NULL);
    if (arrow) {
        XtAddCallback(arrow, XmNexposeCallback, exposeArrow, NULL);
        XtVaSetValues(arrow,
            XmNhighlightThickness,  0,
            XmNshadowThickness,     0);
    }
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app_context);
}

static void
exposeArrow(Widget w, XtPointer client_data, XtPointer call_data)
{
    static GC           gc = NULL;
    XGCValues           values;
    XPoint              points[10];
    Dimension           width, height, shadow;
    Pixel               fg, bg;

    if (XtWindow(w) == NULL) {
        return;
    }
    XtVaGetValues(w, XmNshadowThickness, &shadow,
        XmNforeground,              &fg,
        XmNbackground,              &bg,
        XmNwidth,                   &width,
        XmNheight,                  &height,
```

```
            NULL);
        width -= (2 * shadow);
        height -= (2 * shadow);
        if (!gc) {
            values.fill_style = FillSolid;
            values.foreground = fg;
            values.background = bg;
            gc = XtGetGC(w,
                GCFillStyle | GCForeground | GCBackground,
                &values);
        }
        points[0].x = shadow;
        points[0].y = shadow;
        points[1].x = shadow + (width / 2);
        points[1].y = shadow + height;
        points[2].x = shadow + width;
        points[2].y = shadow;
        points[3].x = shadow;
        points[3].y = shadow;
        XFillPolygon(XtDisplay(w), XtWindow(w), gc, points, 4,
            Convex, CoordModeOrigin);
    }
```

When this program is run, the following widget is produced:



Figure 83 A Combo Box widget containing a custom arrow

The arrow is built by the *exposeArrow()* routine, which is called whenever the custom arrow needs to be drawn on the screen.

Specifying the Arrow Class

By default, the child arrow widget created by Combo Box is an XmArrowButton widget, and is of class `xmArrowButtonWidgetClass`. To create a child arrow widget of a different class, specify the class using the `XmNxrtGearArrowClass` resource. This resource can only be set when the Combo Box is being created.

Modifying the Arrow

You can retrieve the widget ID of the child arrow widget by getting the value of the `XmNxrtGearArrow` resource. This resource is read-only, and cannot be set. Getting this widget ID enables you to change the appearance or behavior of the arrow widget by changing its resources.

9.4.2 Menu Alignment

The `XmNxrtGearAlignment` resource enables you to specify the alignment of the popup menu relative to the text field of the Combo Box. Valid values for this resource, and their effects, are shown in Figure 84.

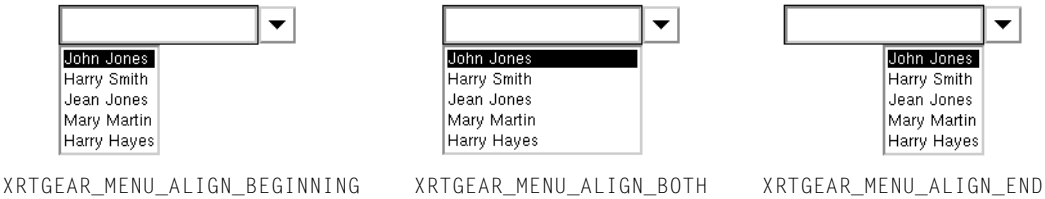


Figure 84 Combo Box menu alignment

By default, `XmNxrtGearAlignment` is set to `XRTGEAR_MENU_ALIGN_BOTH`.

9.4.3 Spacing

By default, there is no space between the text field and the arrow in a Combo Box. To put space between these two child widgets, set the `XmNxrtGearSpacing` resource, as shown in Figure 85.

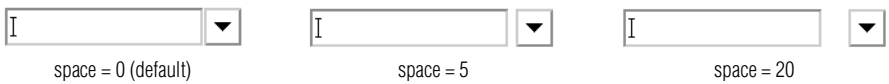


Figure 85 Space between the text field and the arrow

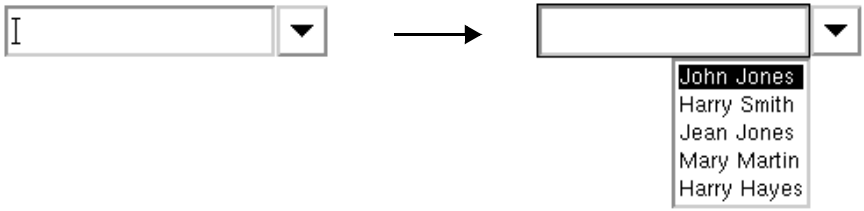
Here, space is measured in pixels.

9.4.4 Number of Items Visible

By default, a maximum of 10 items are displayed in a combo box's drop-down list. This number can be altered using `XmNxrtGearVisibleItemCount`. Whatever the number of visible items, if the picklist count is greater than that number, a vertical scrollbar is displayed.

9.5 User Interaction

Figure 86 illustrates the most common ways users interact with Combo Box widgets. The complete list of translations and actions is provided in section 9.9 on page 255.



MOUSE:
- Click on arrow button to bring up popup menu
- Click on arrow button again to dismiss the popup menu, or press osfCancel or click outside of menu
- Click on item in popup menu to change the value stored in the text field

KEYBOARD:
- Press Down to bring up popup menu
- Move up and down in the popup menu by pressing the Up and Down keys
- Move to a specific item in the popup menu by typing its first character; subsequent keypresses either move to another item or refine the search, depending on the value of `XmNxrtGearSearchPolicy`
- Press Return to select an item from the popup menu
- Press Escape or Ctrl-Up to hide the menu and return the focus to the combo box text field

Figure 86 ComboBox user interactions

Pick List Search Behavior

When the popup menu is displayed, the end-user can search for an item in the pick list by pressing one or more keys. The `XmNxrtGearSearchPolicy` resource controls this popup menu search behavior, and can be set to either `XRTGEAR_SEARCH_SINGLE` (each keypress resets the search, which is the default) or `XRTGEAR_SEARCH_MULTICHAR` (each keypress refines the search).

Figure 87 provides examples that illustrate the behavior of `XmNxrtGearSearchPolicy`.

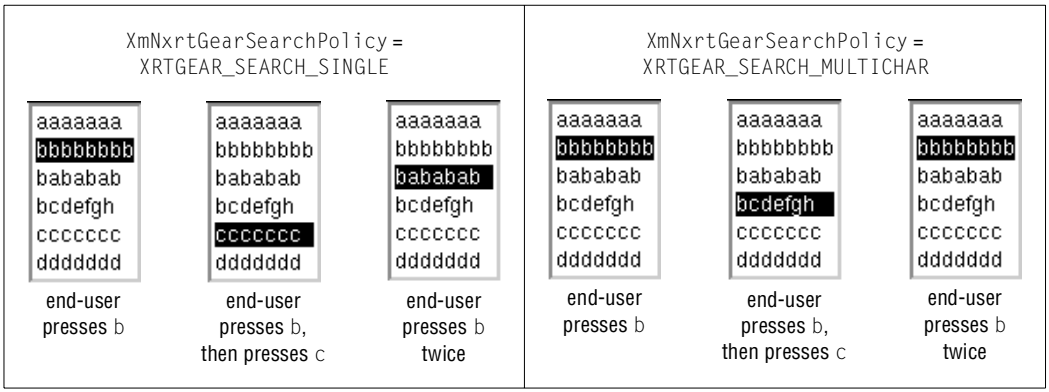


Figure 87 Popup menu search policies

9.6 Auto Matching Completion

The resource `XmNxrtGearAutoComplete`, when `True`, allows auto matching if a pick list has been set. For example, if a picklist contained the values *red*, *green,* and *blue*, then entering "g" while `XmNxrtGearAutoComplete` is `True` jumps to the value *green*, displaying it in the field when the user exits the field and activating the typical field exit callbacks. The characters typed in the field by the user appear in normal foreground and background colors, but the auto-completed portion of the value from the pick list displays in reverse video. `XmNValue` is modified when the user exits the field and contains the full auto-completed value.

9.7 Resource Reference

This section lists all of the resources for the Combo Box widget. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

XmNxrtGearAutoComplete **Boolean** **False** **CSG**

Indicates whether or not to use auto-match completion when a picklist has been set.

Multi-character search is supported. When keys are pressed in a sequence, the current item in the picklist is set to the element whose initial characters match the key sequence. Each keypress refines the search.

XmNxrtGearAlignment **XrtGearMenuAlignment** **XRTGEAR_MENU_ALIGN_BOTH** **CSG**

Specifies the menu alignment relative to the ComboBox text field. Valid values are:

XRTGEAR_MENU_ALIGN_BEGINNING	Align left side of menu with left side of text field
XRTGEAR_MENU_ALIGN_END	Align right side of menu with right side of text field
XRTGEAR_MENU_ALIGN_BOTH	Both sides of menu aligned with text field; menu and text field identical in size

XmNxrtGearArrow **Widget** ***dynamic*** **G**

Specifies the widget ID of the child arrow button. This resource is read-only.

XmNxrtGearArrowClass **WidgetClass** **xmArrowButtonWidgetClass** **C**

Specifies the widget class of the child arrow button. This resource can only be set at creation time.

XmNxrtGearArrowType **XrtGearArrowType** **XRTGEAR_ARROW_WEDGE** **CSG**

Specifies the appearance of the Combo Box arrow. Valid values are:

XRTGEAR_ARROW_NONE	Arrow button displayed with no pixmap
XRTGEAR_ARROW_WEDGE	Arrow contains pixmap similar to those found in Windows 95
XRTGEAR_ARROW_WITH_UNDERBAR	Arrow contains a pixmap similar to those found in Windows 3.1
XRTGEAR_ARROW_CUSTOM	Arrow shape is application-specific

XmNxrtGearMenuCallback **XtCallbackList** **NULL** **CSG**

Specifies the list of callbacks called both before the menu is displayed and before the menu is hidden. Each routine is passed a pointer to an `XrtGearMenuCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an
                             XrtGearMenuCallbackStruct */
)
```

where `XrtGearMenuCallbackStruct` is defined as

```
typedef struct {
    int         reason;         /* read-only */
    XEvent      *event;         /* read-only */
    int         position;       /* read-only */
    String      value;
    Boolean     doit;           /* initially True */
} XrtGearMenuCallbackStruct;
```

reason is either `XRTGEAR_REASON_MENU_BEGIN` (menu about to be displayed) or `XRTGEAR_REASON_MENU_END` (menu about to be hidden). *event* is the event that caused the callback, and is generally a key or button event; if the callback is called programmatically, *event* is `NULL`. *position* is the position of the currently selected menu item in the list; 1 indicates the first menu item, 2 the second, and so on. *value* is only accessible when reason is

XRTGEAR_REASON_MENU_BEGIN; it contains the string that will be displayed in the text field, and can be modified.

The behavior of the *doit* member depends on the value of *reason*. If *reason* is XRTGEAR_REASON_MENU_BEGIN, setting *doit* to False suppresses menu display. If reason is XRTGEAR_REASON_MENU_END, setting *doit* to False tells the Combo Box widget to ignore the end-user's menu item selection.

XmNxrtGearMenuList **Widget** ***dynamic*** **G**

Specifies the widget ID of the pick list (contained in the popup menu). This resource is read-only.

The pick list is of class XmList.

XmNxrtGearPickList **String *** **NULL** **CSG**

Specifies a NULL-terminated list of strings to be displayed in the menu.

XmNxrtGearPickListIndex **int** **XRTGEAR_NOVALUE** **CSG**

Specifies the position of the current item in the list of strings specified by XmNxrtGearPickList. The first item of the list is considered to be at position 0, the second at position 1, and so on. If no item is displayed in the list, this resource is set to XRTGEAR_NOVALUE.

XmNxrtGearSearchPolicy **XrtGearSearchPolicy** **XRTGEAR_SEARCH_SINGLE** **CSG**

Specifies the popup menu search behaviour when the end-user presses one or more keys. Valid values are:

XRTGEAR_SEARCH_SINGLE	Only single-character searching is supported. Each time the end-user presses a key, the current item is set to the first (or next) element whose initial character matches the key. Each keypress restarts the search.
XRTGEAR_SEARCH_MULTICHAR	Multi-character searching is supported. When the end-user presses a sequence of keys, the current item is set to the element whose initial characters match the key sequence. Each keypress refines the search.

XmNxrtGearSpacing **Dimension** **0** **CSG**

Specifies the space in pixels between the text field and the arrow widget.

XmNxrtGearTextField **Widget** ***dynamic*** **G**

Specifies the widget ID of the child field. This resource is read-only.

XmNxrtGearNodeVisibleItemCount **int** **10** **CSG**

Controls the number of items visible in the drop-down list. If there are more items than the specified number, a vertical scrollbar is added.

9.7.1 Resources Inherited by Combo Box

The following resources are inherited by Combo Box from its parent classes. The following table alphabetically lists the resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNinsertPosition	Composite
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNwidth	Core
XmNuserData	XmManager
XmNx	Core
XmNy	Core

9.8 Procedures and Methods Reference

XmCreateXrtGearComboBox()

Creates a Combo Box widget with a field and an arrow button as children. The name of the arrow button is `arrow`, and the name of the field is `field`. The widget ID of the combo box is returned.

```
Widget
XmCreateXrtGearComboBox(
    Widget  parent,
    String  name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent. *name* is the created field's name. *arglist* is a list of *argcount* arguments to be applied to the field. All children are managed.

XmIsXrtGearCombo()

Returns `True` if a widget is a Combo Box widget.

```
int
XmIsXrtGearCombo(
    Widget  widget
    )
```

9.9 Translations and Actions

9.9.1 Translations

Event	Action or Actions
<Key>Cancel	Cancel()
<Key>Escape	Cancel()
<Key>osfHelp	Menu(PrimitiveHelp)
<Key>Return	Menu(ListKbdActivate)
<Key>osfActivate	Menu(ListKbdActivate)
Ctrl <Key>osfDown	Menu(Hide) Menu(TraverseNextTabGroup)
Ctrl <Key>osfUp	Menu(Hide)
Ctrl <Key>osfRight	Cancel() Menu(TraverseRight)
Ctrl <Key>osfLeft	Cancel() Menu(TraverseLeft)
Ctrl <Key>a	Menu(ResetSearch)
Ctrl <Key>n	Menu(FindNext)
Ctrl <Key>p	Menu(FindPrevious)
<Key>osfDown[a]	Menu(Display)
<Key>osfDown[a]	Menu(ListNextItem)
<Key>osfUp	Menu(ListPrevItem)
<Key>osfPageDown	Menu(ListNextPage)
<Key>osfPageUp	Menu(ListPrevPage)
<Key>Tab	Menu(Tab) Menu(TraverseNextTabGroup)
<Key>osfBackSpace	Menu(BackSpace)
<Key>osfDelete	Menu(BackSpace)
<Key>	Menu(Navigate)

a. Pressing osfDown when the pick list is hidden displays the pick list. Pressing osfDown when the pick list is visible moves the highlight to the next item in the list.

9.9.2 Actions

Cancel()
Hides the pick list without selecting an item.

Menu()
Performs a menu operation. This action takes one parameter, which can be any of the following:

PrimitiveHelp	Displays primitive help by calling the help callback associated with the list widget.
ListKbdActivate	Hides the menu. The callbacks in the `XmNxrtGearMenuCallback` list are called. The value of the highlighted item in the list is passed in the *value* member of the callback structure, and then displayed in the field as is.
Display	Displays the menu, and highlights the selected item in the list, if any. If the font list specified by the `XmNfontList` resource contains a font tagged with the "highlight" tag, this font is used to draw the highlighted text.
Hide	Hides the list without changing its state or the field widget state.
ResetSearch	Unhiglights the current item and restores its original font, and moves the first item to the top of the list and highlights it. Also clears the search string.
FindPrevious	Finds the previous string, if any, matching the current search string. If a matching item is found, it is brought into view and highlighted. If no matching item is found, the list remains unchanged.
FindNext	Finds the next string, if any, matching the current search string. If a matching item is found, it is brought into view and highlighted. If no matching item is found, the list remains unchanged.
ListPrevious	Moves the highlight to the previous item in the list.
ListNext	Moves the highlight to the next item in the list.
ListNextPage	Moves the next page into view.
ListPreviousPage	Moves the previous page into view.
Tab	Hides the menu, without changing the field or list widget, and traverses to the mext widget in the tab group.
Navigate	The keystrokes entered by the end-user are concatenated into a search string that is used to navigate to the current or next matching string. If the current key input does not match any string, nothing happens. The match is case-insensitive and anchored at the beginning of the string. When the end of the list is reached, the search starts again from the beginning of the list.
BackSpace	Remove the end-user's last keystroke from the search string.

TraverseLeft	Move to the active item to the left in the widget's active tab group, wrapping around if necessary.
TraverseRight	Move to the active item to the right in the widget's active tab group, wrapping around if necessary.
TraverseNextTabGroup	Make the next tab group in the widget hierarchy the active tab group.

10

The PanedWindow Widget

Sample Program ■ *Widget Synopses*
Resource Reference ■ *Procedures and Methods Reference*
Translations and Actions

10.1 Sample Program

The XrtPanedWindow widget is a composite widget that vertically or horizontally tiles its children. The following is a simple program which creates a PanedWindow widget.

```
#include <Xm/XrtGear.h>
#include <Xm/XrtPanedW.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>

int
#ifndef _NO_PROTO
main(int argc, char *argv[])
#else
main(argc, argv)
int argc;
char *argv[];
#endif
{
    XtAppContext app_context;
     Widget toplevel, form, button, paned;

     /* Initialize Intrinsics */
     toplevel = XtVaAppInitialize(&app_context, "Paned", NULL, 0,
                                  &argc, argv, NULL, NULL);

    form = XtVaCreateManagedWidget("form",
                           xmFormWidgetClass, toplevel,
                           NULL);

    paned = XtVaCreateManagedWidget("paned",
                           xmXrtPanedWindowWidgetClass, form,
                           XmNleftAttachment, XmATTACH_FORM,
                           XmNrightAttachment, XmATTACH_FORM,
```

```
                            XmNtopAttachment, XmATTACH_FORM,
                            XmNbottomAttachment, XmATTACH_FORM,
                            NULL);

    button = XtVaCreateManagedWidget("Button1",
                            xmPushButtonWidgetClass, paned,
                            NULL);
     button = XtVaCreateManagedWidget("Button2",
                            xmPushButtonWidgetClass, paned,
                            NULL);
    button = XtVaCreateManagedWidget("Button3",
                            xmPushButtonWidgetClass, paned,
                            NULL);
    button = XtVaCreateManagedWidget("Button4",
                            xmPushButtonWidgetClass, paned,
                            NULL);

     XtRealizeWidget(toplevel);
     XtAppMainLoop(app_context);

    return(0);
}
```

When this program is compiled and run, the following appears:

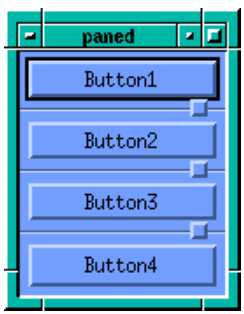


Figure 88 The PanedWindow created by this program

Children appear in a top-to-bottom or left-to-right fashion, with the first child inserted appearing at the top/left of the PanedWindow and the last child inserted appearing in the bottom/right. The PanedWindow grows to match the width of the largest child in vertical mode and the height of its largest child in horizontal mode. All other children are forced to this size. In vertical mode, the height of the PanedWindow is equal to the sum of the heights of all of its children, the spacing between them and the size of the top and bottom margins.

To enable the user to adjust the size of the panes, a pane control sash is created for most children. The sash appears as a small box positioned at the bottom or right of the pane that it controls. The user can adjust the size of the pane by using the mouse or keyboard. See User Interaction on page 261 for more information.

The PanedWindow is a constraint widget, which means it creates and manages a set of contraints for each child. You can specify a minimum and maximum size for each

pane. The PanedWindow does not allow a child to be resized below the minimum or above the maximum size.

Note that when the minimum size is equal to the maximum size, no control sash is displayed.

10.2 Widget Synopses

PanedWindow

Include Files: *<Xm/XrtPanedW.h>*

Class Name: XmXrtPanedWindow

Class Hierarchy: Core → Composite → Constraint →XmManager →XmXrtManager → XmXrtPanedWindow

Class Pointer: XmXrtPanedWindowWidgetClass

Sash

Include Files: *<Xm/XrtSash.h>*

Class Name: XmXrtGearSash

Class Hierarchy: Core → Primitive → XmXrtGearSash

Class Pointer: XmXrtGearSashWidgetClass

Note: The Sash widget is not a public widget and is not accessible by the end user. It can only be created as a child of the XrtPanedWindow.

10.3 User Interaction

Figure 89 illustrates the most common ways users interact with PanedWindow widgets. The complete list of translations and actions is provided in section 10.6 on page 267.



MOUSE:
- Click on sash to give that pane the focus
- Click on sash and drag to resize pane

KEYBOARD:
- Press Tab to move between panes; Shift-Tab to move backwards through panes
- Press osfUp/osfDown/osfLeft/osfRight to resize the pane one line in the desired direction;modify with CTRL to resize pane by one view region for each keypress
- Press space bar to activate children of current pane
- Press Up/Down/Left/Right key to move up/down/left/right through an activated pane's children

Figure 89 PanedWindow user interactions

10.4 Resource Reference

This section lists all of the resources for the PanedWindow widget. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

10.4.1 PanedWindow Resources

XmNxrtGearMarginHeight **Dimension** **3** **CSG**

Specifies the distance between the top and bottom edges of the XrtPanedWindow and its children.

XmNxrtGearMarginWidth **Dimension** **3** **CSG**

Specifies the distance between the left and right edges of the XrtPanedWindow and its children.

XmNxrtGearOrientation **unsigned char** **XmVERTICAL** **CSG**

Specifies the layout as either vertical (with the `XmVERTICAL` value) or horizontal (with the `XmHORIZONTAL` value).

XmNxrtGearPaneResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies the lists of callbacks called before and after each pane is resized due to user action. Each callback is passed an `XrtGearPaneResizeCallbackStruct`:

```
typedef struct {
   XrtGearReason reason;   /* read-only */
   XEvent *event;          /* read-only */
   Widget child;           /* read-only */
   int position;           /* read-only */
   Dimension width;
   Dimension height;
   Boolean doit;
} XrtGearPaneResizeCallbackStruct;
```

reason is either `XRTGEAR_REASON_RESIZE_BEGIN` (pane about to be resized) or `XRTGEAR_REASON_RESIZE_END` (pane has been resized). *event* is the event that caused the callback, and is generally a key or button event. If the callback is called as a result of a programmatic action, *event* is `NULL`. *child* is the Widget being resized and *position* is its position in the list of children of the paned widget. *width* and *height* are the new width and height of the widget and can be modified when *reason* is `XRTGEAR_REASON_RESIZE_BEGIN`. When *reason* is `XRTGEAR_REASON_RESIZE_BEGIN`, setting *doit* to `False` suppresses the resize.

If *width* or *height* are changed when *reason* is `XRTGEAR_REASON_RESIZE_BEGIN`, the new values are used regardless of the values of `XmNxrtGearMinimum` and `XmNxrtGearMaximum`.

XmNxrtGearRecomputeSize **Boolean** **True** **CSG**

If `True` (the default) , the XrtPanedWindow attempts to recompute its size and the sizes of its children when a child is added, removed, or has its size updated. If `False`, the XrtPanedWindow will not update its size.

XmNxrtGearSashHeight **Dimension** **10** **CSG**

Specifies the height of the sash. This value may be adjusted by the XrtPanedWindow if the sash is too large to display.

XmNxrtGearSashIndent **Position** **-10** **CSG**

Specifies the horizontal indent of the sash along each plane. A positive value indents the sash from the left/top of the XrtPanedWindow, and a negative value indents the sash from the bottom/right of the XrtPanedWindow. If the offset is greater than the width of the XrtPanedWindow, the sash is placed flush against the edge.

XmNxrtGearSashShadowThickness **Dimension** ***dynamic*** **CSG**

Specifies the thickness fo the shadows of the sashes.

XmNxrtGearSashWidth **Dimension** **10** **CSG**

Specifies the width of the sash. This value may be adjusted by the XrtPanedWindow if the sash is too large to display.

XmNxrtGearSeparatorShow **Boolean** **True** **CSG**

Determines whether a separator is created between each of the panes.

XmNxrtGearSpacing **Dimension** **8** **CSG**

Specifies the distance between each child pane.

10.4.2 PanedWindow Constraint Resources

XmNxrtGearAllowResize **Boolean** **False** **CSG**

Allows an application to specify whether the XrtPanedWindow allows a pane to request to be resized. This flag has an effect only after the XrtPanedWindow and its children have been realized. If this flag is set to `True`, the XrtPanedWindow tries to honor requests to alter the height/width of the pane. If `False`, it always denies pane requests to resize.

XmNxrtGearMinimum **Dimension** **1** **CSG**

Specifies the minimum size of the pane. This value must be less than or equal to the maximum.

XmNxrtGearMaximum **Dimension** **1000** **CSG**

Specifies the maximum size of the pane. This value must be greater than or equal to the minimum.

XmNxrtGearPosition **int** **XmLAST_POSITION** **CSG**

Specifies the position of the widget in its parent's list of children (the list of pane children, not including the sashes). The value is an int no less than 0 and no greater than the number of children in the list at the time the value is specified. A value of 0 means that the child is placed at the beginning of the list. The value can also be specified as `XmLAST_POSITION` (the default), which means that the child is placed at the end of the list. Any other value is ignored. `XtGetValues` returns the position of the widget in its parent's child list at the time of the call to `XtGetValues`.

When a widget is inserted into its parent's child list, the positions of any existing children that are greater than the specified widget's `XmNxrtGearPosition` are increased by one. The effect of an `XtSetValues` for `XmNxrtGearPosition` is to remove the specified widget from its parent's child list, decrease by one the positions of any children that are greater than the specified widgets' former position in the list, and then insert the widget into the parent's child list as described above.

XmNxrtGearSkipAdjust **Boolean** **False** **CSG**

Allows the application to specify that the XrtPanedWindow should not automatically resize the pane when set to `True`.

10.4.3 Resources Inherited by PanedWindow

The following resources are inherited by PanedWindow from its parent classes. The following table alphabetically lists the resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNbackgroundPixmap	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	XmManager
XmNbottomShadowPixmap	XmManager
XmNchildren	Composite
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	XmManager
XmNheight	Core
XmNhelpCallback	XmManager
XmNhighlightColor	XmManager
XmNhighlightPixmap	XmManager

Resource	Inherited From
XmNinitialFocus	XmManager
XmNinitialResourcesPersistent	Core
XmNmappedWhenManaged	Core
XmNnavigationType	XmManager
XmNnumChildren	Composite
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	XmManager
XmNstringDirection	XmManager
XmNtopShadowColor	XmManager
XmNtopShadowPixmap	XmManager
XmNtranslations	Core
XmNtraversalOn	XmManager
XmNunitType	XmManager
XmNuserData	XmManager
XmNwidth	Core
XmNx	Core
XmNy	Core

10.5 Procedures and Methods Reference

XmCreateXrtPanedWindow()

Creates an XrtPanedWindow widget with the specified parent, name and arguments.

```
Widget
XmCreateXrtPanedWindow(
    Widget  parent,
    String  name,
    Arg     *arglist
    int     argcount
    )
```

parent is the parent; *name* is the created PanedWindow's name; *arglist* is a list of *argcount* arguments to be applied to the PanedWindow. All children are managed.

XmIsXrtPanedWindow()

Returns `True` if a widget is a PanedWindow widget or a subclass of XrtPanedWindow.

```
Boolean
XmIsXrtPanedWindow(
    Widget  widget,
    )
```

XrtPanedWindowGetPaneList()

Returns the XrtList containing pointers (of type `Widget *`) to all the children of the paned window that are panes (not separators or sashes). This list will never be `NULL`, but could contain zero elements.

```
XrtGearObject
XrtPanedWindowGetPaneList(
    Widget  widget,
    )
```

10.6 Translations and Actions

10.6.1 Translations

Event	Action or Actions
~Ctrl ~Shift ~Meta ~Alt <Btn1Down>	SashAction(Start)
~Ctrl ~Shift ~Meta ~Alt <Btn1Motion>	SashAction(Move)
~Ctrl ~Shift ~Meta ~Alt <Btn1Up>	SashAction(Commit)
~Ctrl ~Shift ~Meta ~Alt <Btn2Down>	SashAction(Start)
~Ctrl ~Shift ~Meta ~Alt <Btn2Motion>	SashAction(Move)
~Ctrl ~Shift ~Meta ~Alt <Btn2Up>	SashAction(Commit)
<Key>osfActivate	PrimitiveParentActivate()
<Key>osfCancel	PrimitiveParentCancel()
<Key>osfHelp	Help()
Ctrl <Key>osfUp	SashAction(Key,Large,Up)
<Key>osfUp	SashAction(Key,Default,Up)
Ctrl <Key>osfRight	SashAction(Key,Large,RIght)
<Key>osfRight	SashAction(Key,Default,RIght)
Ctrl <Key>osfDown	SashAction(Key,Large,Down)
<Key>osfDown	SashAction(Key,Default,Down)
Ctrl <Key>osfLeft	SashAction(Key,Large,Left)
<Key>osfLeft	SashAction(Key,Default,Left)
~Shift ~Meta ~Alt <Key>Return	PrimitiveParentActivate()
Shift ~Meta ~Alt <Key>Tab	PrevTabGroup()
~Meta ~Alt <Key>osfTab	NextTabGroup()

10.6.2 Actions

Help()

Calls the callbacks for the `XmNhelpCallback`, if any exist. If there are no help callbacks for this widget, this action calls the help callbacks for the nearest widget ancestor that does have help callbacks.

NextTabGroup()

Moves the keyboard focus to the next tab group. By default, each pane and sash is a tab group.

PrevTabGroup()

Moves the keyboard focus to the previous tab group. By default, each pane and sash is a tab group.

SashAction(start)

Activates the interactive placement of the pane's borders.

SashAction(move)

Causes the sash to track the position of the cursor. If one of the panes reaches its maximum or minimum size, adjustment continues with the next adjustable pane.

SashAction(commit)

Ends the sash movement.

SashAction(key, increment, direction)

When the sash action is caused by a keyboard event, the sash with the keyboard focus is moved according to the *increment* and *direction* specified. The default is to adjust the sash by one line. **Large** adjusts the sash by one view region. The *direction* is specified as any of `Up`, `Down`, `Left` or `Right`.

Note that *SashAction()* is not a direct action of the XrtPanedWindow, but is an action of the Sash control created by the XrtPanedWindow.

11

The Widget Tips Utility

Introduction ■ *Enabling Widget Tips*
Adding Widget Tips to a Widget ■ *Retrieving the Widget List*
Help Balloon Appearance ■ *Display Callbacks*
Display Delay ■ *Disabling Widget Tips*
Manually Displaying and Removing Help Text ■ *Defining Help Translations*
Resource Reference ■ *Procedures and Methods Reference*

11.1 Introduction

The Widget Tips utility allows the end-user to view a temporary help balloon for a widget by resting the cursor on it. The text in the balloon is displayed in a rectangular window near the cursor, which is erased after the cursor is moved from the window or a key/button is pressed.

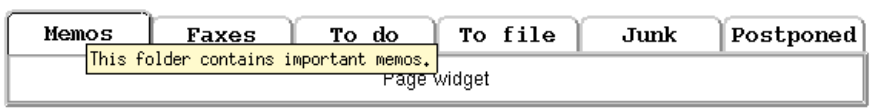


Figure 90 A widget with its Widget Tips help text displayed

To provide a help balloon for any widget, add Widget Tips resources to it. Widget Tips resources can be set on a widget from a resource file or via the *XrtGearTipsVaSetValues()* or *XrtGearTipsSetValues()* methods; *XtVaSetValues()* cannot be used.

11.2 Enabling Widget Tips

To enable the Widget Tips utility for a particular shell widget and its children, call the *XrtGearTipsEnable()* method, as shown below:

```
#include <Xm/XrtTips.h>

Widget toplevel;

XrtGearTipsEnable(toplevel, True);
```

The first argument to *XrtGearTipsEnable()* can be any widget within the shell widget for your application. The second argument is a Boolean indicating whether to enable or disable Widget Tips; to disable, set this argument to `False`.

You must include the file *Xm/XrtTips.h* to be able to use Widget Tips.

If a child is added to the shell after *XrtGearTipsEnable()* is called, Widget Tips will not be enabled on the new widget. (*XrtGearTipsEnable()* must be called again to enable Widget Tips on this new child.)

11.3 Adding Widget Tips to a Widget

To add Widget Tips to a widget, set the `XmNxrtGearText` resource by calling *XrtGearTipsVaSetValues()*:

```
XmString str;

str = XmStringCreateLtoR("my help text",
        XmSTRING_DEFAULT_CHARSET);
XrtGearTipsVaSetValues(widget,
    XmNxrtGearText,         str,
    NULL);
```

Alternatively, you can use *XrtGearTipsSetValues()*:

```
ArgList args;
Cardinal num_args;

XrtGearTipsSetValues(widget, args, num_args);
```

You cannot set `XmNxrtGearText` or any other Widget Tips resource using *XtVaSetValues()* or *XtSetValues()*.

`XmNxrtGearText` expects a value of type XmString. If `XmNxrtGearText` is set to `NULL` (the default value), no help balloon is displayed.

Getting Widget Tips Resources

To retrieve the value of any Widget Tips resource, use *XrtGearTipsVaGetValues()* or *XrtGearTipsGetValues()*. These methods are identical in behavior to the Motif methods *XtVaGetValues()* and *XtGetValues()*.

11.4 Retrieving the Widget List

To retrieve a list of the child widgets of a shell, call the *XrtGearTipsGetWidgetList()* method:

```
Widget toplevel, *list;
int num_widgets;
Boolean enabled_only = False;
Boolean retrieved;

retrieved = XrtGearTipsGetWidgetList(toplevel,
    enabled_only, &list, &num_widgets);
```

XrtGearTipsGetWidgetList() retrieves a pointer to a list of the child widgets of the shell, and the number of widgets in this list. Using this list, you can enable Widget Tips for some or all of the widgets in your application.

To return a list of the widgets for which Widget Tips has already been enabled, set the second argument, *enabled_only*, to `True`.

XrtGearTipsGetWidgetList() returns `True` if the retrieve was successful.

11.5 Help Balloon Appearance

The following resources control the appearance of the Widget Tips help balloons:

- The foreground and background colors of the help balloon are specified by the `XmNxrtGearForeground` and `XmNxrtGearBackground` resources. These resources are of type Pixel.
- The `XmNxrtGearFontList` resource specifies the fontlist to use when displaying the help text.
- The `XmNxrtGearTipsBorderType` resource specifies the border style to use for the help text border. The valid values of this resource are illustrated in Figure 91. The default value is `XRTGEAR_BORDER_PLAIN`.

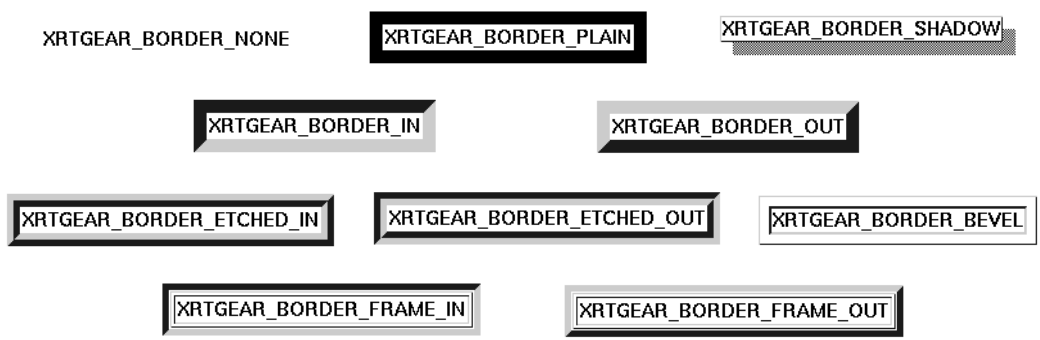


Figure 91 Widget Tips borders

Each of these resources must be set by calling *XrtGearTipsVaSetValues()*, not *XtVaSetValues()*. The following example code sets these resources:

```
XrtGearTipsVaSetValues(widget,
    XtVaTypedArg, XmNxrtGearForeground,
        XmRString, "red", strlen("red")+1,
    XtVaTypedArg, XmNxrtGearBackground,
        XmRString, "white", strlen("white")+1,
    XtVaTypedArg, XmNxrtGearFontList,
        XmRString, "-*-times-medium-r-*-*-*-120-*-*-*-*-iso8859-1",
        strlen("-*-times-medium-r-*-*-*-120-*-*-*-*-iso8859-1")+1,
    XmNxrtGearTipsBorderType, XRTGEAR_BORDER_ETCHED_IN,
    NULL);
```

11.6 Display Callbacks

The `XmNxrtGearDisplayCallback` resource allows you to change the help text or suppress help display. Each callback in this list is called both before and after the help balloon appears.

Each callback is passed a pointer to the following structure:

```
typedef struct {
    int         reason;     // Read-only
    XEvent      *event;     // Read-only
    Widget      widget;     // Read-only
    Boolean     doit;       // initially True
    XmString    text;
} XrtGearDisplayCallbackStruct;
```

reason is either `XRTGEAR_REASON_DISPLAY_BEGIN` (before help text display) or `XRTGEAR_REASON_DISPLAY_END` (after display). When a callback is invoked before help text display, you can change the text to be displayed by changing *text*[1], or suppress help display by setting *doit* to `False`.

1. Before changing the text, you must first free the existing help text using *XmStringFree()*.

For example, the following code suppresses help display if a global variable named `SuppressHelpBalloons` is set:

```
int         SuppressHelpBalloons;

void
displayCB(Widget w, XtPointer client_data,
XrtGearDisplayCallbackStruct *call_data)
{
    if (call_data->reason == XRTGEAR_REASON_DISPLAY_BEGIN) {
        if (SuppressHelpBalloons) {
            call_data->doit = False;
        }
    }
}

int
main(int argc, char **argv)
{
    Widget  w;
    ...
    XrtGearTipsAddCallback(w, XmNxrtGearDisplayCallback,
        displayCB, NULL);
    ...
}
```

Note that the *XrtGearTipsAddCallback()* method, not *XtAddCallback()*, must be used to add callbacks to the `XrtGearDisplayCallbackStruct` list. Similarly, *XrtGearTipsRemoveCallback()* must be used to delete a callback from this list.

11.7 Display Delay

The `XmNxrtGearDelay` resource specifies the time, in milliseconds, that the cursor must remain stationary inside the widget before the help balloon is displayed. For example, the following code sets the delay time to 1000 milliseconds (one second):

```
XrtGearTipsVaSetValues(widget,
    XmNxrtGearDelay,        1000,
    NULL);
```

By default, `XmNxrtGearDelay` is set to 500 milliseconds.

Help Display Groups

Normally, when focus changes from one widget to another, the delay time specified by `XmNxrtGearDelay` must elapse before the help balloon is displayed for the new widget. To override this, you can specify that groups of widgets are to be treated as a single *help display group*. When the end-user moves from one widget in a help display group to another in the same group, `XmNxrtGearDelay` is ignored and the help balloon in the second widget is displayed immediately.

To define a help display group, pick an arbitrary positive integer value, and set the XmNxrtGearGroup resource for each widget in the group to this value. For example, the following code defines two help display groups numbered 1 and 2:

```
XmString str, str2, str3;

str = XmStringCreateLocalized(
    "This folder contains important memos.");
XrtGearTipsVaSetValues(memos,
    XmNxrtGearText,     str,
    XmNxrtGearGroup,    1,
    NULL);
str2 = XmStringCreateLocalized("This string appears immediately.");
XrtGearTipsVaSetValues(faxes,
    XmNxrtGearText,     str2,
    XmNxrtGearGroup,    1,
    NULL);
str3 = XmStringCreateLocalized("This string appears later.");
XrtGearTipsVaSetValues(todo,
    XmNxrtGearText,     str3,
    XmNxrtGearGroup,    2,
    NULL);
```

When the end-user moves from the *memos* widget to the *faxes* widget, the help text for *faxes* is displayed immediately, since *memos* and *faxes* are in the same help display group. When the end-user then moves to *todo*, he or she will have to wait for the delay time to elapse before the help text for *todo* appears.

Any of the following special values can be used when setting XmNxrtGearGroup:

XRTGEAR_PARENT	This help display group consists of all widgets which are children of a particular parent (except for the children whose display group number has been explicitly set to something else).
XRTGEAR_SELF	This indicates that the widget does not belong to any help display group.

By default, XmNxrtGearGroup is XRTGEAR_PARENT.

11.8 Disabling Widget Tips

The following code shows how to disable Widget Tips for a particular widget:

```
XrtGearTipsVaSetValues(widget,
    XmNxrtGearDelay,    MAXINT,
    XmNxrtGearGroup,    XRTGEAR_SELF,
    NULL);
```

Setting XmNxrtGearDelay to a very large number effectively disables the help balloon display, as the end-user is not likely to keep the cursor in place long enough to display the help. Setting XmNxrtGearGroup to XRTGEAR_SELF ensures that this widget is not part of a help display group (which causes the help text to display immediately regardless of the value of XmNxrtGearDelay if the delay has been previously triggered).

11.9 Manually Displaying and Removing Help Text

Displaying Help

The *XrtGearTipsDisplayHelp()* method enables you to display the specified text manually (regardless of whether Widget Tips has been enabled). The text can appear at any place on the screen.

The following Enhanced Toggle state changed callback procedure uses *XrtGearTipsDisplayHelp()* to display the number of times a button has been pressed:

```
int click_count = 0;
XmString str;

void
stateCB(Widget w, XtPointer client_data,
    XrtGearStateChangedCallbackStruct *call_data)
{
    char buf[80];

    if (call_data->reason != XRTGEAR_REASON_STATE_BEGIN) {
        return;
    }
    if (click_count != 0) {
        XmStringFree(str);
    }
    click_count++;
    sprintf(buf, "number of clicks: %d", click_count);
    str = XmStringCreateSimple(buf);
    XrtGearTipsDisplayHelp(w, False, 10, 10, str);
}
```

XrtGearTipsDisplayHelp() takes the following arguments:

- The widget for which the help text is to be displayed
- A Boolean indicating whether the location of the help text is relative to the root window (`True`) or the widget (`False`)
- The X-coordinate of the help text position
- The Y-coordinate of the help text position
- The text to display (if `NULL`, display the help text previously defined by setting the `XmNxrtGearText` resource)

Help text displayed by *XrtGearTipsDisplayHelp()* is not automatically removed when the pointer leaves the widget. It must be removed manually using *XrtGearTipsRemoveHelp()*, described below.

Removing Help

To remove the help balloons displayed by *XrtGearTipsDisplayHelp()*, call *XrtGearTipsRemoveHelp()*:

```
XrtGearTipsRemoveHelp();
```

This procedure removes all currently displayed help balloons.

11.10 Defining Help Translations

You can specify that a Widget Tips help balloon is to be activated when the end-user presses a key (or generates any other specified X event). To do this, set the event to call the *XrtGearTipsActionDisplay()* action routine.

You can also specify an event that removes a displayed help balloon. To do this, set the event to call the *XrtGearTipsActionRemove()* action routine.

For example, the following code specifies that a help balloon is to appear whenever an end-user presses Shift and the middle mouse button, and disappear when the end-user releases the button:

```
char translations[] = "\
    Shift<Btn2Down>:    tips_display_action()\n\
    Shift<Btn2Up>:      tips_remove_action()";

XtActionsRec actions[] = {
    {"tips_display_action",
        (XtActionProc) XrtGearTipsActionDisplay},
    {"tips_remove_action",
        (XtActionProc) XrtGearTipsActionRemove},
};

Widget  toplevel;
Widget  *widget_list;
int     num_widgets;
int     i;

XtAppAddActions(XtWidgetToApplicationContext(toplevel),
    actions, XtNumber(actions));
XrtGearTipsGetWidgetList(toplevel, False, &widget_list,
    &num_widgets);
for (i = 0; i < num_widgets; i++) {
    XtOverrideTranslations(widget_list[i],
        XtParseTranslationTable(translations));
}
```

The call to *XrtGearTipsGetWidgetList()* ensures that the translation is to be specified for each widget in the application.

11.11 Resource Reference

This section lists all of the resources for the Widget Tips utility. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

XmNxrtGearAlignment **unsigned char** **XmALIGNMENT_CENTER** **CSG**

Specifies the position of the text within the window: `XmALIGNMENT_BEGINNING`, `XmALIGNMENT_CENTER` or `XmALIGNMENT_END`.

XmNxrtGearBackground **Pixel** ***see below*** **CSG**

Specifies the help balloon's background color. The default is the Pixel equivalent of Lemon Chiffon.

XmNxrtGearDelay **int** **500** **CSG**

Specifies the time (in milliseconds) that the cursor must remain stationary inside the widget before help is displayed. This delay is `0` if the cursor is next moved into a widget with the same `XmNxrtGearGroup`.

XmNxrtGearDisplayCallback **XtCallbackList** **NULL** **CSG**

Specifies the list of callbacks called before and after help is displayed. The routine must be added and removed using *XrtGearTipsAddCallback()* and *XrtGearTipsRemoveCallback()*. Each routine is passed a pointer to an `XrtGearDisplayCallbackStruct`. Add callbacks to the list using the *XtAddCallback()* method. The form of the callback is:

```
void
CallbackProcedure(
    Widget      widget,
    XtPointer   client_data,
    XtPointer   call_data /* points to an XrtGearDisplayCallbackStruct */
)
```

where `XrtGearDisplayCallbackStruct` is defined as:

```
typedef struct {
    int         reason;     // Read-only
    XEvent      *event;     // Read-only
    Widget      widget;     // Read-only
    Boolean     doit;       // Initially True
    XmString    text;
} XrtGearDisplayCallbackStruct;
```

reason is `XRTGEAR_REASON_DISPLAY_BEGIN` or `XRTGEAR_REASON_DISPLAY_END`; *event* is `NULL` if the help is brought up manually using *XrtGearTipsDisplayHelp()*; *widget* is the widget for which help is displayed; *doit* should be set to `False` if *reason* is `XRTGEAR_REASON_DISPLAY_BEGIN` and the help is not to be displayed; *text* is the text to be displayed, and may be changed if *reason* is `XRTGEAR_REASON_DISPLAY_BEGIN` (the current text must first be freed using *XmStringFree()*).

XmNxrtGearFontList **XmFontList** **fixed** **CSG**

Specifies the fontlist in which to display the text.

XmNxrtGearForeground **Pixel** **Black** **CSG**

Specifies the help balloon's foreground color.

XmNxrtGearGroup **int** **XRTGEAR_PARENT** **CSG**

Specifies an arbitrary number assigned to the widget, used by `XmNxrtGearDelay` processing. This number must be a positive integer.

The following special values are defined for `XmNxrtGearGroup`:

`XRTGEAR_PARENT`	This help display group consists of all widgets which are children of a particular parent (except for the children whose display group number has been explicitly set to something else). This is the default.

XRTGEAR_SELF	This indicates that the widget does not belong to any help display group.	

XmNxrtGearShadowThickness **Dimension** **1** **CSG**

Specifies the help balloon's shadow thickness.

XmNxrtGearText **XmString** **NULL** **CSG**

Specifies the text to be displayed. If NULL, no help balloon is displayed.

XmNxrtGearTipsBorderType **XrtGearBorderType** **XRTGEAR_BORDER_PLAIN** **CSG**

Specifies the style used for the window border. See Enhanced Label's description.

11.12 Procedures and Methods Reference

This section lists the Widget Tips procedures and methods in alphabetical order.

XrtGearTipsActionDisplay()

Action routine which displays the help balloon for a widget.

```
void
XrtGearTipsActionDisplay(
    Widget          widget,
    XEvent          *event
    )
```

widget is the widget for which help is to be displayed. *event* is the X event that triggers the action routine.

XrtGearTipsActionRemove()

Action routine which removes the displayed help balloon for a widget.

```
void
XrtGearTipsActionRemove(
    Widget          widget,
    XEvent          *event
    )
```

widget is the widget for which help is to be displayed. *event* is the X event that triggers the action routine.

XrtGearTipsAddCallback()

Must be used to add the XmNxrtGearDisplayCallback callback.

```
void
XrtGearTipsAddCallback(
    Widget          widget,
    String          callback_name,
    XtCallbackProc  proc,
    XtPointer       tag
    )
```

widget is the widget for which the callback is being defined; *callback_name* must be `XmNxrtGearDisplayCallback`; *proc* is the new callback being added; *tag* is a pointer to the argument to be passed to the callback procedure when it is invoked.

XrtGearTipsAddCallbacks()

Adds a list of `XmNxrtGearDisplayCallback` callbacks.

```
void
XrtGearTipsAddCallbacks(
    Widget          widget,
    String          callback_name,
    XtCallbackList  callbacks
    )
```

widget is the widget for which the new callbacks are being defined. *callback_name* must be `XmNxrtGearDisplayCallback`. *callbacks* is the list of callbacks to be added.

XrtGearTipsDisplayHelp()

Displays help for a widget, or allows text to be displayed at any point on the screen.

```
void
XrtGearTipsDisplayHelp(
    Widget          widget,
    Boolean         position_root,
    Position        x, y,
    XmString        text
    )
```

widget is the widget for which help is to be displayed. If *position_root* is `True`, *x* and *y* are the help position relative to the root window; if *position_root* is `False`, *x* and *y* are the help position relative to the widget. *text* is the text written if no help text is found for the widget.

XrtGearTipsEnable()

Enables or disables the display of help for all widgets in a shell. Any widget within the desired shell may be passed.

```
void
XrtGearTipsEnable(
    Widget          shell,
    Boolean         enable
    )
```

shell is the shell widget (or any widget within the shell); *enable* is either `True` (enable help) or `False` (disable help).

If a child is added to the shell after *XrtGearTipsEnable()* is called, Widget Tips will not be enabled on the new widget. (*XrtGearTipsEnable()* must be called again to enable Widget Tips on this new child.)

XrtGearTipsGetValues()

Gets Widget Tips resources for a widget.

```
void
XrtGearTipsGetValues(
    Widget          widget,
    ArgList         args,
    Cardinal        num_args
    )
```

This method's arguments are similar to those passed to *XtGetValues()*.

XrtGearTipsGetWidgetList()

Retrieves either a list of all widgets which are children of a shell, or a list of only those widgets in a shell for which Widget Tips is enabled. Returns `True` if the retrieve was successful.

```
Boolean
XrtGearTipsGetWidgetList(
    Widget          shell,
    Boolean         enabled_only,
    Widget          **list,
    int             *num_widgets
    )
```

shell is the shell widget. *list* contains the list of returned widgets; this list must be freed after use. If *enabled_only* is `True`, *list* contains only those widgets for which Widget Tips is enabled; if *enabled_only* is `False`, *list* contains a list of all child widgets. *num_widgets* returns the number of elements in *list*.

XrtGearTipsRemoveAllCallbacks()

Must be used (in place of *XtRemoveAllCallbacks()*) to remove all the `XmNxrtGearDisplayCallback` callbacks.

```
void
XrtGearTipsRemoveAllCallbacks(
    Widget          widget,
    String          callback_name
    )
```

widget is the widget whose callbacks are to be removed; *callback_name* must be `XmNxrtGearDisplayCallback`.

XrtGearTipsRemoveCallback()

Must be used (in place of *XtRemoveCallback()*) to remove the `XmNxrtGearDisplayCallback` callback.

```
void
XrtGearTipsRemoveCallback(
    Widget          widget,
    String          callback_name,
    XtCallbackProc  proc,
    XtPointer       tag
    )
```

widget is the widget containing the callback to be removed; *callback_name* is `XmNxrtGearDisplayCallback`; *proc* is the callback procedure to be removed; *tag* is a

pointer to the client data to match with the registered callback procedure. The callback is only removed if *proc* and *tag* match a callback in the list.

XrtGearTipsRemoveCallbacks()

Removes a list of `XmNxrtGearDisplayCallback` callbacks.

```
void
XrtGearTipsRemoveCallbacks(
    Widget          widget;
    String          callback_name,
    XtCallbackList  callbacks
    )
```

widget is the widget containing the callbacks to be removed; *callback_name* must be `XmNxrtGearDisplayCallback`. *callbacks* is the list of callbacks to be removed.

XrtGearTipsRemoveHelp()

Removes any help currently displayed.

```
void
XrtGearTipsRemoveHelp(void)
```

XrtGearTipsSetValues()

Sets Widget Tips resources (including `XmNxrtGearDisplayCallback`) on a widget.

```
void
XrtGearTipsSetValues(
    Widget          widget,
    ArgList         args,
    Cardinal        num_args
    )
```

This method's arguments are similar to those passed to *XtSetValues()*.

XrtGearTipsTrackingEvent()

Similar to *XmTrackingEvent()*, this routine displays help for the widget next clicked by the end-user within the shell. The selected widget is returned.

```
Widget
XrtGearTipsTrackingEvent(
    Widget          shell,
    Cursor          cursor,
    Boolean         confine_to,
    XEvent          *event_return
    )
```

shell is the shell widget; *cursor* specifies the cursor to be used as a pointer; *confine_to* specifies whether the pointer is confined to *shell. event_return* is the returned `ButtonRelease` or `KeyRelease` event.

XrtGearTipsVaGetValues()

Gets Widget Tips resources for a widget.

```
void
XrtGearTipsVaGetValues(
    Widget  widget,
    ...
    NULL
    )
```

This method's arguments are similar to those passed to *XtVaGetValues()*.

XrtGearTipsVaSetValues()

Sets Widget Tips resources (including `XmNxrtGearDisplayCallback`) on a widget.

```
void
XrtGearTipsVaSetValues(
    Widget  widget,
    ...
    NULL
    )
```

This method's arguments are similar to those passed to *XtVaSetValues()*.

12

The Widget Snapshot Utility

Introduction ■ *Pixmap Snapshots*
Writing and Printing a Pixmap ■ *XImage Snapshots*
Resource Reference ■ *Procedures and Methods Reference*

12.1 Introduction

The Widget Snapshot utility enables you to print the contents of any Motif widget, including Manager widgets, in PostScript format. Pixmap and XImage images of any Motif widget can also be obtained.

12.2 Pixmap Snapshots

To obtain a pixmap snapshot of a widget, call the *XrtGearSnapGetWidgetPixmap()* method:

```
Pixmap p;

p = XrtGearSnapGetWidgetPixmap(widget);
```

Any Motif widget can be snapped in this way.

The pixmap returned by *XrtGearSnapGetWidgetPixmap()* should be freed after use; to do this, call *XFreePixmap()*.

12.3 Writing and Printing a Pixmap

To convert a pixmap to PostScript format and append it to a file, call the *XrtGearSnapVaPrintPixmap()* method:

```
Pixmap p;
FILE *file;

file = fopen("mypixmap.ps", "w");
XrtGearSnapVaPrintPixmap(widget, p, file, NULL);
fclose(file);
```

The file can now be printed just like any other standard PostScript file.

The widget passed to *XrtGearSnapVaPrintPixmap()* can be any widget in the display. *XrtGearSnapVaPrintPixmap()* returns a Boolean value indicating whether the write operation was successful. The file is assumed to be already open. Any pixmap can be passed to this method, including, but not limited to, the pixmaps created by calling *XrtGearSnapGetWidgetPixmap()*.

Specifying Print Attributes

You can specify one or more print attribute-value pairs when calling *XrtGearSnapVaPrintPixmap()*. For example, the following invocation specifies two copies of the pixmap are to be written to the file:

```
XrtGearSnapVaPrintPixmap(widget, p, file,
    XRTGEAR_SNAP_NUM_COPIES, 2,
    NULL);
```

A call to *XrtGearSnapVaPrintPixmap()* can supply any or all of the following attributes (and their values):

`XRTGEAR_SNAP_COLOR`	color specification: one of `XRTGEAR_SNAP_AS_IS` (default), `XRTGEAR_SNAP_MONO` or `XRTGEAR_SNAP_COLOR`
`XRTGEAR_SNAP_MARGIN_BOTTOM`	bottom margin; must be a floating-point number of type *double*; default is 0.25 units (as defined by `XRTGEAR_SNAP_UNITS`)
`XRTGEAR_SNAP_MARGIN_LEFT`	left margin; must be a floating-point number of type *double*; default is 0.25 units
`XRTGEAR_SNAP_MARGIN_RIGHT`	right margin; must be a floating-point number of type *double*; default is 0.25 units
`XRTGEAR_SNAP_MARGIN_TOP`	top margin; must be a floating-point number of type *double*; default is 0.25 units
`XRTGEAR_SNAP_PAPERSIZE_WIDTH`	paper width; must be a floating-point number of type *double*; default is 8.5 units
`XRTGEAR_SNAP_PAPERSIZE_HEIGHT`	paper width; must be a floating-point number of type *double*; default is 11.0 units

XRTGEAR_SNAP_UNITS	unit of measurement for the above; must be either XRTGEAR_SNAP_INCHES (default) or XRTGEAR_SNAP_CM
XRTGEAR_SNAP_NUM_COPIES	number of copies; default is 1
XRTGEAR_SNAP_ORIENTATION	widget orientation on page; one of XRTGEAR_SNAP_PORTRAIT (default), XRTGEAR_SNAP_LANDSCAPE or XRTGEAR_SNAP_ BEST
XRTGEAR_SNAP_SCALE	page scale to use; one of XRTGEAR_SNAP_FIT_TO_PAGE, XRTGEAR_SNAP_FIT_TO_PAGE_WIDTH, XRTGEAR_SNAP_FIT_TO_PAGE_HEIGHT or 72.0 units (default)
XRTGEAR_SNAP_SHOWPAGE	default is True

The *XrtGearSnapPrintPixmap()* method enables you to supply attribute-value pairs as an ArgList:

```
XrtGearSnapArg *arglist;
int numargs;

XrtGearSnapPrintPixmap(widget, p, file, arglist,
        numargs);
```


12.4 XImage Snapshots

The *XrtGearSnapGetWidgetImage()* method obtains an XImage snapshot of a widget:

```
XImage *ximage;

ximage = XrtGearSnapGetWidgetImage(widget);
```

Any Motif widget can be snapped in this way.

The XImage returned by *XrtGearSnapGetWidgetImage()* should be freed after use; to do this, call *XDestroyImage()*.

By linking in the PDS utils library, you can save this XImage in xpm or xwd format using the function *XrtGearSnapSaveImage()*. You can also pass your own image functions to *XrtGearSnapSaveImage()* and save the image in any format you need to support.

12.5 Resource Reference

This section lists all of the resources for the Widget Snapshot utility. Listed after the resource name are its data type, default value and a list of the procedures which may be used with the resource. **C** means it can be defined in a create (e.g. *XtCreateWidget()*

), **S** means it can be set (e.g. *XtSetValues()*), and **G** means it can be used in a get (e.g. *XtGetValues()*).

XmNxrtGearSnapNoShapeClip **Boolean** **False** **CSG**

Turn off shape clipping in the *XrtGearSnapGetWidgetPixmap()* method. (Set this resource in a resource file or .Xdefaults file if your server does not handle shape clipping properly.)

12.6 Procedures and Methods Reference

This section lists the Widget Snapshot procedures and methods in alphabetical order.

XrtGearSnapGetWidgetImage()

Gets an XImage snapshot of a widget, returning `NULL` if the widget is not managed or realized. The image should be freed after use with *XDestroyImage()*.

```
XImage *
XrtGearSnapGetWidgetImage(
    Widget  widget
    )
```

widget is the widget to be snapped.

XrtGearSnapGetWidgetPixmap()

Gets a pixmap snapshot of a widget, returning `XmUNSPECIFIED_PIXMAP` if the widget is not managed or realized. The pixmap should be freed after use with *XFreePixmap()*.

```
Pixmap
XrtGearSnapGetWidgetPixmap(
    Widget  widget
    )
```

widget is the widget to be snapped.

XrtGearSnapPrintPixmap()

Appends an encapsulated PostScript snapshot of a pixmap to a file, returning `False` if the file could not be written to. (The file is assumed to be already open.)

```
Boolean
XrtGearSnapVaPrintPixmap(
    Widget          widget,
    Pixmap          pixmap,
    FILE            *fp,
    XrtGearSnapArg  *args,
    int             num_args,
    NULL
    )
```

widget is any widget on the display; *pixmap* is the pixmap snapshot; *fp* is the file to contain the PostScript output; *args* is a list of attribute-value pairs (see *XrtGearSnapVaPrintPixmap()* for a description); *num_args* is the number of arguments in the list specified by *args*.

XrtGearSnapSaveImage()

Saves snapshot of a pixmap to a file, returning False if the file could not be written to. (The file is assumed to be already open.)

```
Boolean
XrtGearSnapSaveImage(
    Widget      widget,
    XtPointer   funcs,
    String      filename,
    String      msg
);
```

widget is any widget on the display; *funcs* is a pointer to an XrtPDSImageFunc; *filename* is the name of the file to save the image to; *msg* is the buffer where any error messages are stored. If *msg* is `NULL`, the error message is printed to the console.

XrtGearSnapVaPrintPixmap()

Appends an encapsulated PostScript snapshot of a pixmap to a file, returning `False` if the file could not be written to. (The file is assumed to be already open.)

```
Boolean
XrtGearSnapVaPrintPixmap(
    Widget  widget,
    Pixmap  pixmap,
    FILE    *fp,
    ...
    NULL
    )
```

This method is passed three mandatory arguments, plus an optional list of attribute-value pairs. *widget* is any widget on the display; *pixmap* is the pixmap snapshot; *fp* is the file to contain the PostScript output.

Attributes which can be specified are (default values are in italics):

XRTGEAR_SNAP_COLOR	*XRTGEAR_SNAP_AS_IS*, XRTGEAR_SNAP_MONO, XRTGEAR_SNAP_COLOR
XRTGEAR_SNAP_MARGIN_BOTTOM	0.25
XRTGEAR_SNAP_MARGIN_LEFT	0.25
XRTGEAR_SNAP_MARGIN_RIGHT	0.25
XRTGEAR_SNAP_MARGIN_TOP	0.25
XRTGEAR_SNAP_NUM_COPIES	1
XRTGEAR_SNAP_ORIENTATION	*XRTGEAR_SNAP_PORTRAIT*, XRTGEAR_SNAP_LANDSCAPE, XRTGEAR_SNAP_BEST
XRTGEAR_SNAP_PAPERSIZE_WIDTH	8.5
XRTGEAR_SNAP_PAPERSIZE_HEIGHT	11.0
XRTGEAR_SNAP_SCALE	XRTGEAR_SNAP_FIT_TO_PAGE, XRTGEAR_SNAP_FIT_TO_PAGE_WIDTH, XRTGEAR_SNAP_FIT_TO_PAGE_HEIGHT, *72.0*
XRTGEAR_SNAP_SHOWPAGE	True

XRTGEAR_SNAP_UNITS	*XRTGEAR_SNAP_INCHES*, XRTGEAR_SNAP_CM

The attributes XRTGEAR_SNAP_MARGIN_LEFT, XRTGEAR_SNAP_MARGIN_RIGHT, XRTGEAR_SNAP_MARGIN_TOP, XRTGEAR_SNAP_MARGIN_BOTTOM, XRTGEAR_SNAP_PAPERSIZE_WIDTH and XRTGEAR_SNAP_PAPERSIZE_HEIGHT must have their values specified as floating-point numbers. All floating-point values are in the units specified by XRTGEAR_SNAP_UNITS.

13

Lists, Strings, Colors and Icons

The XrtList Object ■ *The XrtString Object*
Icons ■ *The XrtColorEditor Object*
Procedures and Methods Reference

XRT/gear provides the following objects and data types for use in its widgets:

- The XrtList object, which stores groups of items of the same size
- The XrtString object, which stores Strings, XmStrings or icons
- The XrtColorEditor object, which allows the developer to quickly insert a smart color editor
- The XrtGearIcon data type, which specifies an icon that appears in an XRT/gear widget

This chapter lists the procedures and methods that manipulate XrtList and XrtString objects, and the procedures and methods that create XrtColorEditor objects and XrtGearIcon structures.

13.1 The XrtList Object

The XrtList object provides a convenient way of storing groups of items of the same size. Items can be added, deleted, sorted, and retrieved using methods defined for these purposes. The following sections describe these methods.

Note that the list specified by an XrtList object is zero-based: the first element has index 0, the second element has index 1, and so on.

13.1.1 Sample Code

The following is a sample procedure that shows how to create a list, add items to it, and retrieve an item.

```
void
CreateListOfIntegers()
{
    XrtGearObject list;
    int i, *ptr;

    list = XrtGearListCreate(sizeof(int));
    i = 34;
    XrtGearListAppend(list, &i);
    i = 46;
    XrtGearListAppend(list, &i);
    ptr = XrtGearListGetItem(list, 1);
    fprintf (stderr, "item is %d\n", *ptr);
}
```

This procedure creates a list of integers, and adds the integers 34 and 46 to the list. It then retrieves the second element of the list, and displays it.

13.1.2 Creating and Destroying a List

To create a list, call the *XrtGearListCreate()* method:

```
list = XrtGearListCreate(sizeof(int));
```

This method is passed one parameter, which is the size of each of the elements in the list. (Elements of different types can be stored in the same list, provided they are the same size.) The created list is returned.

To destroy a list created by *XrtGearListCreate()*, call *XrtGearListDestroy()*:

```
XrtGearListDestroy(list);
```

All memory used by the list is freed.

Duplicating a List

To make a copy of an existing list, use the *XrtGearListDuplicate()* method:

```
newlist = XrtGearListDuplicate(oldlist);
```

This makes a copy of the list and all its elements, and returns a pointer to the copy. The original list is unchanged.

13.1.3 Adding and Deleting List Items

Adding List Items

You can add an item to an existing list in any of four places:

- At the end of a list
- Immediately preceding an existing item
- Immediately following an existing item
- In its proper (sorted order) position

To add an item to the end of an existing list, call *XrtGearListAppend()*:

```
index = XrtGearListAppend(list, newitem);
```

This method requires two parameters: the list to be added to, and the item to be added. This item must be the same size as the size specified when the list was created by *XrtGearListCreate()* . The method returns the index of the new item (with 0 representing the first element of the list, 1 the second, and so on).

To place a new item in front of an existing list item, call *XrtGearListInsertBefore()*:

```
newindex = XrtGearListInsertBefore(list, oldindex, newitem);
```

XrtGearListInsertBefore() requires three parameters: the list to be added to, the index of the existing list item (with 0 representing the first item), and the item to be added. The index of the new item is returned.

Similarly, *XrtGearListInsertAfter()* places a new item immediately after an existing list item:

```
newindex = XrtGearListInsertAfter(list, oldindex, newitem);
```

Its parameters and return value are identical to those of *XrtGearListInsertBefore()*.

If your list is sorted, you can add an item in its proper sorted position by calling *XrtGearListAdd()*:

```
index = XrtGearListAdd(list, newitem);
```

This method requires two parameters: the list and the item to be added. The index of the new item is returned. (For information on sorting lists, see section 13.1.6 on page 296.)

Removing and Deleting List Items

The XrtList object defines routines that remove an item from a list, and additional routines that both remove an item from a list and destroy the item.

To remove an item from a list, call *XrtGearListRemove()*:

```
retpos = XrtGearListRemove(list, position);
```

This method is passed the list and the position of the item to be removed. This position is also returned.

To remove all items from a list, call *XrtGearListRemoveAll()*:

```
XrtGearListRemoveAll(list);
```

Note that neither of the above methods actually destroys the item memory. To destroy an item as well as remove it, call *XrtGearListDelete()*:

```
retpos = XrtGearListDestroy(list, position);
```

This method, like *XrtGearListRemove()*, is passed the list and the position of the item to be removed, and returns the position.

To destroy all items in a list, call *XrtGearListDestroyAll()*:

```
XrtGearListDestroyAll();
```

If you have defined additional data structures associated with a particular item, which need to be destroyed when the item is deleted, you can define a destroy procedure by calling *XrtGearListSetDestroyProc()*:

```
XrtGearListSetDestroyProc(list, my_destroy_proc);
```

The destroy procedure is called whenever *XrtGearListDestroy()* or *XrtGearListDestroyAll()* is called, and must be defined to be of the following type:

```
void my_destroy_proc (
    XrtGearObject       object,
    XtPointer           item
    );
```

object is the list; *item* is the item to be destroyed.

The XrtList object also enables you to destroy an object in a list by calling *XrtGearListDestroyObject()*:

```
XrtGearObject           list, *objptr;

XrtGearListDestroyObject(list, objptr);
```

The two parameters are the list itself and a pointer to the object to be destroyed.

To destroy a non-object item in a list, call *XrtGearListDestroyXtPointer()*:

```
XrtGearObject           list;
XtPointer               *item;

XrtGearListDestroyXtPointer(list, item);
```

This method is passed the list and a pointer to the item to be destroyed.

13.1.4 Searching and Retrieving List Items

The XrtList object defines several methods that provide the following search and retrieval operations:

- Determining whether an XrtGearObject is a list
- Finding an item in a list
- Retrieving an indexed element of a list or the entire array of elements in the list
- Searching forward or backward in a list

Other methods enable you to obtain the number of items in the list and their size.

Checking a List

The *XrtGearListIsList()* method determines whether an XrtGearObject is actually an XrtList:

```
Boolean                 is_list;
XrtGearObject           object;

is_list = XrtGearListIsList(object);
```

XrtGearListIsList() returns `True` if the object is an XrtList object, and `False` if it is not.

Finding an Item in a List

To find an item in a list, use either of the following methods:

- *XrtGearListFind()*, which finds a list item matching a specified item using the XrtList comparison procedure
- *XrtGearListLookupItem()*, which finds a list item matching a specified item by comparing the corresponding memory

Both methods are passed the list and the item to be compared. Both return the index of the matched item if found, or `XRTGEAR_LIST_ITEM_NOT_FOUND` if the item is not contained in the list.

To specify the comparison procedure to be used by *XrtGearListFind()*, call *XrtGearListSetCompareProc()*:

```
XrtGearListSetCompareProc(list, my_compare_proc);
```

This comparison procedure must be defined to have the following prototype:

```
int my_compare_proc (
    XtPointer           item1,
    XtPointer           item2
    );
```

item1 and *item2* are the two items to be compared. The comparison procedure must return the following:

- a positive integer if *item1* is greater than *item2*
- 0 if the two items are equal
- a negative integer if *item1* is less than *item2*

To retrieve the comparison procedure currently in use, call *XrtGearListGetCompareProc()*:

```
XrtGearListItemCompareProc myproc;

myproc = XrtGearListGetCompareProc(list);
```

The returned procedure is of type `XrtGearListItemCompareProc`, which is defined to be the following:

```
typedef int (*XrtGearListItemCompareProc) (
    XtPointer item1, XtPointer item2
);
```

This comparison procedure is also used in sorting. See section 13.1.6 on page 296 for a description of list sorting.

Retrieving an Item By Index

To retrieve an item given its position, call *XrtGearListGetItem()*:

```
XtPointer               item;
XrtGearObject           list;
int                     index;

item = XrtGearListGetItem(list, index);
```

XrtGearListGetItem() is passed two parameters: the list, and the position of the item to be returned (with 0 representing the first item, 1 the second, and so on). The item at this position is returned. If the position is invalid, `NULL` is returned.

To set the item at a particular position, call *XrtGearListSetItem()*:

```
XrtGearListSetItem(list, index, item);
```

This method is passed three arguments: the list, the index of the new item (with 0 representing the first item), and the new item itself. This method returns `True` if the set operation was successful, or `False` if the index specified is invalid.

Retrieving the List Array

To retrieve the entire array of list elements, call *XrtGearListGetItems()*:

```
XrtGearObject           list;
XtPointer               array;

array = XrtGearListGetItems(list);
```

If the list is empty, `NULL` is returned.

Searching for an Item

To search forward in a list for a specified item, call *XrtGearListSearchForward()*:

```
int                     pos;
XrtGearObject           list;
int                     start_position;
XtPointer               item;

pos = XrtGearListSearchForward(list, start_position, item);
```

This method requires three parameters: the list, the position in the list at which to start searching (0 represents the first item), and the item to search for. If the item is matched, the position of the item in the list is returned; otherwise, `XRTGEAR_LIST_ITEM_NOT_FOUND` is returned.

To search backward in a list for a specified item, call *XrtGearListSearchBackward()*:

```
pos = XrtGearListSearchBackward(list, start_position, item);
```

The parameters and return value are the same as those of *XrtGearListSearchForward()*.

Retrieving List Information

To obtain the number of items in a list, call *XrtGearListGetItemCount()*:

```
int                     numitems;
XrtGearObject           list;

numitems = XrtGearListGetItemCount(list);
```

This method is passed the list as its argument.

To obtain the size of an item in the list, call *XrtGearListGetItemSize()*:

```
int                     itemsize;
XrtGearObject           list;

itemsize = XrtGearListGetItemSize(list);
```

13.1.5 List Comparison Routines

The XrtList object provides a variety of routines that compare values of various types. Routines are defined which compare double-precision floating point numbers, single-precision floating-point numbers, integers and strings. All of the routines defined in this section can be the comparison procedure used by *XrtGearListFind()*; to use one of these routines as the comparison procedure, pass it to *XrtGearListSetCompareProc()*.

To compare two double-precision floating-point numbers, call *XrtGearCmpDoubles()*:

```
XtPointer           double1, double2;
int                 retval;

retval = XrtGearCmpDoubles(double1, double2);
```

This routine returns 1 if the first number is larger, 0 if they are equal, and -1 if the second number is larger.

XrtGearCmpFloats() compares two single-precision floating-point numbers:

```
XtPointer           float1, float2;
int                 retval;

retval = XrtGearCmpFloats(float1, float2);
```

The return value is the same as that of *XrtGearCmpDoubles()*.

To compare two integers, call *XrtGearCmpInts()*:

```
XtPointer           int1, int2;
int                 retval;

retval = XrtGearCmpInts(int1, int2);
```

The return value is the same as that of *XrtGearCmpDoubles()*.

To compare two strings, call *XrtGearCmpStrings()*:

```
XtPointer           string1, string2;
int                 retval;

retval = XrtGearCmpStrings(string1, string2);
```

This routine returns 1 if the first string is alphabetically greater than the second string, 0 if they are equal, and -1 if the second string is alphabetically greater.

XrtGearCmpStrings() is case-sensitive. To perform a case-insensitive comparison of two strings, call *XrtGearCmpStringsIgnoreCase()*:

```
XtPointer           string1, string2;
int                 retval;

retval = XrtGearCmpStringsIgnoreCase(string1, string2);
```

The return value is the same as that of *XrtGearCmpStrings()*.

13.1.6 Sorting a List

To sort the items in a list, call *XrtGearListSort()*:

```
XrtGearListSort(list);
```

The comparison procedure defined by *XrtGearListSetCompareProc()* is used to determine the sort order. If no comparison procedure is specified, the following default sort behavior is used:

- If the list items are of a size less than or equal to 8, the items are sorted in byte order
- If the size is greater than 8, no sorting is performed if no comparison procedure is defined.

For a description of *XrtGearListSetCompareProc()*, see section 13.1.4 on page 292.

To determine whether a list is sorted, call *XrtGearListIsSorted()*:

```
int                 is_sorted;

is_sorted = XrtGearListIsSorted(list);
```

This method returns a non-zero value if the list is sorted, and 0 if it is not.

Relocating List Elements

To move list elements from one location in the list to another, call *XrtGearListMoveItemsBefore()*:

```
Boolean             ok;
XrtGearObject       list;
int                 destination;
int                 source;
int                 num_items;

source = 3;         /* fourth element is the first to be moved */
num_items = 3;      /* move three elements */
destination = 0;    /* place before first element */
ok = XrtGearListMoveItemsBefore(list, destination, source,
    num_items);
```

XrtGearListMoveItemsBefore() requires four parameters:

- The list containing the items to be moved
- The index of the destination element (with 0 representing the first element); the block of elements being moved is placed immediately before this element
- The index of the first element of the block being moved
- The number of elements to be moved

XrtGearListMoveItemsBefore() returns `True` if the move was successful, or `False` if an error occurred.

13.1.7 User-Defined Data

The XrtList object enables you to define user data to be associated with the list. To define user data, pass a pointer to the user data to the *XrtGearListSetUserData()* method:

```
XrtGearObject       list;
XtPointer           user_data;

XrtGearListSetUserData(list, user_data);
```

You are responsible for freeing the memory used by the user data.

To retrieve the current user data associated with a list, call *XrtGearListGetUserData()*:

```
user_data = XrtGearListGetUserData(list);
```

If no user data is defined for this list, *XrtGearListGetUserData()* returns `NULL`.

13.2 The XrtString Object

The XrtString object enables you to use one kind of data object to store character strings, XmStrings and icons. The following sections describe the methods that manipulate XrtStrings.

13.2.1 Creating and Destroying a String

To create an XrtString object, use the *XrtGearStringCreate()* method:

```
XrtGearStringObjectClass    class;
XtPointer                   stringval;
XrtGearObject               newstring;

newstring = XrtGearStringCreate(class, stringval);
```

The created string is returned; it is of type `XrtGearObject`.

The class parameter must be one of the following:

- `XrtGearCharStringObjectClass` (create a string)
- `XrtGearXmStringObjectClass` (create an XmString)
- `XrtGearIconStringObjectClass` (create an icon)

For example, the following call creates an XrtString object of class `XrtGearCharStringObjectClass`:

```
XrtGearStringCreate(XrtGearCharStringObjectClass, "mystring");
```

The *XrtGearStringCreateCharString()*, *XrtGearStringCreateXmString()* and *XrtGearStringCreateIconString()* methods provide convenient ways of creating XrtStrings of any class:

```
XrtGearObject           charclass, xmclass;
XmString                xmstring;
XrtGearIcon             icon;

charclass = XrtGearStringCreateCharString("mystring");
xmclass = XrtGearStringCreateXmString(xmstring);
iconclass = XrtGearStringCreateIconString(&icon);
```

For details on creating XrtGearIcons, see section 13.4 on page 300.

To destroy an XrtString object, call *XrtGearStringDestroy()*:

```
XrtGearObject           mystring;

XrtGearStringDestroy(mystring);
```

All memory allocated to the string is freed at the time it is destroyed.

Copying a String

To copy an existing string, call the *XmNxrtGearStringCopy()* method:

```
XrtGearObject           original, mycopy;

mycopy = XmNxrtGearStringCopy(original);
```

13.2.2 Determining the String Type

To determine whether an object of type XrtGearObject is an XrtString, and its class if it is a string, call *XrtGearStringIsCharString()*, *XrtGearStringIsXmString()* or *XrtGearStringIsIconString()*:

```
Boolean                 is_string;
XrtGearObject           object;

is_string = XrtGearStringIsCharString(object);
is_string = XrtGearStringIsXmString(object);
is_string = XrtGearStringIsIconString(object);
```

These methods behave as follows:

- *XrtGearStringIsCharString()* returns `True` if the object it is passed is an XrtString of class `XrtGearCharStringObjectClass`.
- *XrtGearStringIsXmString()* returns `True` if the object is an XrtString of class `XrtGearXmStringObjectClass`.
- *XrtGearStringIsIconString()* returns `True` if the object is an XrtString of class `XrtGearIconStringObjectClass`.

To test whether an XrtGearObject is an XrtList object, use *XrtGearListIsList()*, described in section 13.1.4 on page 292.

13.2.3 Manipulating the String Value

To get the value of an XrtString object, call *XrtGearStringGetValue()*:

```
XrtGearObject           object;
XtPointer               value;

value = XrtGearStringGetValue(object);
```

This method behaves as follows:

- If the object is of class `XrtGearCharStringObjectClass`, the value returned is of type String
- If the object is of class `XrtGearXmStringObjectClass`, the value returned is of type XmString.
- If the object is of class `XrtGearIconStringObjectClass`, the value returned is a pointer to an XrtGearIcon.

To set the current value of an XrtString object, call *XrtGearStringSetValue()*:

```
XrtGearStringSetValue(object, value);
```

When setting XrtStrings using string values, make sure that the value is of the correct type. If the XrtString is of class `XrtGearCharStringObjectClass`, the new value must be of type String. If the XrtString is of class `XrtGearXmStringObjectClass`, the new value must be of type XmString. The following code example illustrates this:

```
if (XrtGearStringIsCharString(object)) {
    XrtGearStringSetValue(object, "mynewvalue");
} else if (XrtGearStringIsXmString(object)) {
    XrtGearStringSetValue(object,
        XmStringCreateSimple("mynewvalue"));
}
```

13.2.4 Comparing Two Strings

To compare the values of two strings, call *XrtGearStringCompare()*:

```
int                 result;
XrtGearObject       object1, object2;

result = XrtGearStringCompare(object1, object2);
```

The returned value is one of the following:

- an integer greater than zero if the first value is alphanumerically greater than the second
- 0 if the two values are identical
- a negative integer if the first value is alphanumerically less than the second

Each of the XrtStrings can be of either string class (`XrtGearCharStringObjectClass` or `XrtGearXmStringObjectClass`). Objects of different string classes can be compared.

13.3 The XrtColorEditor Object

The XrtColorEditor object allows a developer to quickly insert a smart color editor into an application with very little coding. This color editor allows the user to select a color by name, hexidecimal string, RGB value or HSB (Hue, Saturation, Brightness) value. The object takes care of all user interaction, and allocating and previewing any selected colors.

13.3.1 Sample Code

1. Creating a color editor:

```
Widget parent;
XrtGearObject editor;
editor = XrtGearColorEditorCreate(parent, "Test Editor");
```

2. Testing the editor:

```
XrtGearObject editor;
if (XrtGearIsColorEditor(editor)) {
   printf("\"editor\" *is* an XrtColorEditor\n");
}
else {
   printf("\"editor\" is *not* an XrtColorEditor\n");
}
```

3. Destroying the editor:

```
XrtGearObject editor:
XrtGearColorEditorDestroy(editor);
```

4. Getting the current red green and blue values:

```
XrtGearObject editor;
unsigned int red, green, blue;
XrtGearColorEditorGetRGBValues(editor, &red, &green, &blue);
printf("Get RGB: red = %d, green = %d, blue = %d\n", red, green, blue);
```

5. Setting a color by its HSB values:

```
XrtGearColorEditorSetHSBValues(editor, 300, 80, 80);
```

Section 13.5.3 lists the methods you can use to manipulate the XrtColorEditor.

13.4 Icons

The XrtGearIcon data type enables you to store an icon's pixmap and its pixmap mask in one structure for convenience. It is defined as follows:

```
typedef struct {
    Pixmap          icon;
    Pixmap          icon_mask;
} XrtGearIcon;
```

The following routines create and manipulate XrtGearIcon structures:

- *XrtGearIconLoadFromXpmData()* reads an XrtGearIcon structure from an XPM data structure.
- *XrtGearIconLoadFromXpmFile()* reads an XrtGearIcon structure from a file in XPM format.

- *XrtGearIconGetSize()* returns the width and height of the specified icon.

See section 13.5.4 on page 319 for a description of these methods. Refer to the XPM Manual for more details on the XPM format.

13.5 Procedures and Methods Reference

13.5.1 XrtList Object Methods

XrtGearCmpDoubles()

Compares two double-precision floating-point numbers; returns 1 if the first one is larger, 0 if they are equal, and -1 if the second one is larger. This procedure is suitable for passing to *XrtGearListSetCompareProc()*.

```
int
XrtGearCmpDoubles(
    XtPointer              double1,
    XtPointer              double2
    )
```

double1 and *double2* are the two numbers to be compared.

XrtGearCmpFloats()

Compares two (single-precision) floating-point numbers; returns 1 if the first one is larger, 0 if they are equal, and -1 if the second one is larger. This procedure is suitable for passing to *XrtGearListSetCompareProc()*.

```
int
XrtGearCmpFloats(
    XtPointer       float1,
    XtPointer       float2
    )
```

float1 and *float2* are the two numbers to be compared.

XrtGearCmpInts()

Compares two integers; returns 1 if the first one is larger, 0 if they are equal, and -1 if the second one is larger. This procedure is suitable for passing to *XrtGearListSetCompareProc()*.

```
int
XrtGearCmpInts(
    XtPointer       int1,
    XtPointer       int2
    )
```

int1 and *int2* are the two numbers to be compared.

XrtGearCmpStrings()

Compares two strings; returns 1 if the first one is (alphabetically) larger, 0 if they are equal, and -1 if the second one is larger. This procedure is suitable for passing to *XrtGearListSetCompareProc()*.

```
int
XrtGearCmpStrings(
    XtPointer       string1,
    XtPointer       string2
    )
```

string1 and *string2* are the two strings to be compared.

XrtGearCmpStringsIgnoreCase()

Compares two strings in case-insensitive fashion; returns 1 if the first one is (alphabetically) larger, 0 if they are equal, and -1 if the second one is larger. This procedure is suitable for passing to *XrtGearListSetCompareProc()*.

```
int
XrtGearCmpStringsIgnoreCase(
    XtPointer       string1,
    XtPointer       string2
    )
```

string1 and *string2* are the two strings to be compared.

XrtGearListAdd()

Adds an item to the list in its proper (sorted) position. The sort order is defined by the procedure returned by *XrtGearListGetCompareProc()*. Returns the position index of the new node, or `XRTGEAR_LIST_INVALID_POSITION` on failure.

```
int
XrtGearListAdd(
    XrtGearObject   list,
    XtPointer       new
    )
```

list is a list object. *new* is a pointer to the new item, which must be the same size as the value returned by *XrtGearListGetItemSize()*.

XrtGearListAppend()

Appends an item to the end of a list. Returns the position index of the new node, or `XRTGEAR_LIST_INVALID_POSITION` on failure.

```
int
XrtGearListAppend(
    XrtGearObject   list,
    XtPointer       new
    )
```

list is a list object. *new* is a pointer to the new item, which must be the same size as the value returned by *XrtGearListGetItemSize()*.

XrtGearListCreate()

Creates a zero-length list of items, and returns it.

```
XrtGearObject
XrtGearListCreate(
    size_t          item_size
    )
```

item_size is the intended size of the items in the list.

XrtGearListDelete()

Deletes the item at the specified position. Returns the position of the deleted item.

```
int
XrtGearListDelete(
    XrtGearObject   list,
    int             position
    )
```

list is a list object; *position* is the position of the item to be deleted.

If a destroy procedure has been registered (by calling *XrtGearSetDestroyProc()*), it is called when the item is deleted. (To remove an item without calling the destroy procedure, use *XrtGearListRemove()*.)

XrtGearListDeleteAll()

Deletes all the items from the list.

```
void
XrtGearListDeleteAll(
    XrtGearObject   list,
    )
```

list is a list object whose items are to be deleted.

If a destroy procedure has been registered (by calling *XrtGearSetDestroyProc()*), it is called each time an item is deleted. (To remove the items without calling the destroy procedure, use *XrtGearListRemoveAll()*.)

XrtGearListDestroy()

Destroys the specified list and frees all its allocated memory.

```
void
XrtGearListDestroy(
    XrtGearObject   list
    )
```

list is a list object to be destroyed.

XrtGearListDestroyObject()

Destroys an object in the specified list.

```
void
XrtGearListDestroyObject(
    XrtGearObject   list,
    XrtGearObject   *object
    )
```

list is a list object containing the object to be destroyed; *object* is the object to be destroyed.

XrtGearListDestroyXtPointer()

Destroys a non-object item in the specified list.

```
void
XrtGearListDestroyXtPointer(
    XrtGearObject   list,
    XtPointer       *item
    )
```

list is a list object containing the item to be destroyed; *item* is the item to be destroyed.

XrtGearListDuplicate()

Makes an exact copy of a list and returns the copy. Note: The new list's item destroy procedure is set to `NULL`; it can be set using *XrtGearListSetDestroyProc()*.

```
XrtGearObject
XrtGearListDuplicate(
    XrtGearObject   list
    )
```

list is a list object to be copied.

XrtGearListFind()

Finds a list item matching a specified item (using the procedure returned by *XrtGearListGetCompareProc()* to check for equality). Returns the position index of the item if found, or `XRTGEAR_LIST_ITEM_NOT_FOUND` if the item is not found.

```
int
XrtGearListFind(
    XrtGearObject   list,
    XtPointer       item
    )
```

list is a list object to search; *item* is a pointer to the item to match. If the list is sorted (i.e., if `XmNxrtGearSorted` is set to `True`), a binary search is used to look up the item.

If there are duplicate items in the list, the item returned is the first of the duplicates.

XrtGearListGetCompareProc()

Retrieves the procedure that the object uses for searching and sorting.

```
XrtGearListItemCompareProc
XrtGearListGetCompareProc(
    XrtGearObject       list
    )
```

list is a list object whose compare procedure is being returned.

XrtGearListGetItem()

Returns a pointer to the list item at a specified index. Returns NULL if the index is invalid. You should dereference the returned item.

```
XtPointer
XrtGearListGetItem(
    XrtGearObject       list,
    int                 index
    )
```

list is a list object; *index* is the index of the desired item, with 0 representing the first element of the list, 1 the second, and so on.

XrtGearListGetItemCount()

Retrieves the number of items stored in a list.

```
int
XrtGearListGetItemCount(
    XrtGearObject       list
    )
```

list is a list object whose items are to be counted.

XrtGearListGetItems()

Returns a pointer to the array of items in the list object, or NULL if the list is empty.

```
XtPointer
XrtGearListGetItems(
    XrtGearObject       list
    )
```

list is a list object.

XrtGearListGetItemSize()

Returns the size of each of the individual items in alist.

```
int
XrtGearListGetItemSize(
    XrtGearObject       list
    )
```

list is a list object whose item size is being returned.

XrtGearListGetUserData()

Gets the user data associated with an XrtList object and returns it.

```
XtPointer
XrtGearListGetUserData(
    XrtGearObject   list
    )
```

list is a list object whose user data is to be retrieved.

XrtGearListInsertAfter()

Inserts a new item into a list after a specified position. Returns the index of the new node, or `XRTGEAR_LIST_INVALID_POSITION` on failure.

```
int
XrtGearListInsertAfter(
    XrtGearObject   list,
    int             index,
    XtPointer       new
    )
```

list is a list object; *index* is the position after which the new item is to be added; *new* is a pointer to the new item. If *index* is greater than the number of elements in the list, *new* is appended to the end of the list.

Note that *new* must be the same size as the value returned by *XrtGearListGetItemSize()*.

XrtGearListInsertBefore()

Inserts a new item into a list before a specified position. Returns the index of the new node, or `XRTGEAR_LIST_INVALID_POSITION` on failure.

```
int
XrtGearListInsertBefore(
    XrtGearObject   list,
    int             index,
    XtPointer       new
    )
```

list is a list object; *index* is the position after which the new item is to be added; *new* is a pointer to the new item. If *index* is greater than the number of elements in the list, *new* is appended to the end of the list.

Note that *new* must be the same size as the value returned by *XrtGearListGetItemSize()*.

XrtGearListIsList()

Returns `True` if the XrtGearObject being passed is an XrtList object.

```
Boolean
XrtGearListIsList(
    XrtGearObject   object
    )
```

object is an object to be checked.

XrtGearListIsSorted()

Indicates whether the list is currently in a sorted state. Returns a non-zero value if the list is sorted.

```
int
XrtGearListIsSorted(
    XrtGearObject           list
    )
```

list is a list object to be checked.

XrtGearListLookupItem()

Locates the item in the specified list. This routine performs a strict memory compare and does *not* use the procedure returned by *XrtGearListGetCompareProc()*. Returns the position of the matched item, or XRTGEAR_LIST_ITEM_NOT_FOUND on failure.

```
int
XrtGearListLookupItem(
    XrtGearObject           list,
    XtPointer               item
    )
```

list is a list to be searched; *item* is a pointer to the item to be located.

XrtGearListMoveItemsAfter()

Moves a specified number of items from a source location to after the destination location.

```
Boolean
XrtGearListMoveItemsAfter(
    XrtGearObject           list,
    int                     destination,
    int                     source,
    int                     num_items_to_move
    )
```

list is a list object. *destination* is the destination location; the moved items are placed after this location. *source* is the location of the first item to move. *num_items_to_move* is the number of items to be moved.

XrtGearListMoveItemsBefore()

Moves a specified number of items from a source location to before the destination location.

```
Boolean
XrtGearListMoveItemsBefore(
    XrtGearObject           list,
    int                     destination,
    int                     source,
    int                     num_items_to_move
    )
```

list is a list object. *destination* is the destination location; the moved items are placed before this location. *source* is the location of the first item to move. *num_items_to_move* is the number of items to be moved.

XrtGearListRemove()

Removes the item at the specified position from a list. Returns the position of the removed item.

```
int
XrtGearListRemove(
    XrtGearObject   list,
    int             position
    )
```

list is a list object; *position* is the position of the item to be removed.

The destroy procedure (registered by calling *XrtGearSetDestroyProc()*), is *not* called when the item is removed. (To delete an item and call the destroy procedure, use *XrtGearListDelete()*.)

XrtGearListRemoveAll()

Removes all the items from a list.

```
void
XrtGearListRemoveAll(
    XrtGearObject   list
    )
```

list is a list object whose items are to be removed.

The destroy procedure (registered by calling *XrtGearSetDestroyProc()*), is *not* called when the item is removed. (To delete each item, calling the destroy procedure each time, use *XrtGearListDeleteAll()*.)

XrtGearListSearchForward()

Searches forward in a list, from a specified location, for a match for a specified item. (The procedure returned by *XrtGearListGetCompareProc()* is used to perform comparisons.) Returns the position of the matched item, or `XRTGEAR_LIST_ITEM_NOT_FOUND` on failure.

```
int
XrtGearListSearchForward(
    XrtGearObject           list,
    int                     start_position,
    XtPointer               item
    )
```

list is a list to be searched; *start_position* is the position of the first list item to be compared; *item* is a pointer to the item to be matched.

XrtGearListSearchReverse()

Searches backward in a list, from a specified location, for a match for a specified item. (The procedure returned by *XrtGearListGetCompareProc()* is used to perform comparisons.) Returns the position of the matched item, or `XRTGEAR_LIST_ITEM_NOT_FOUND` on failure.

```
int
XrtGearListSearchReverse(
    XrtGearObject           list,
    int                     start_position,
    XtPointer               item
    )
```

list is a list to be searched; *start_position* is the position of the first list item to be compared; *item* is a pointer to the item to be matched.

XrtGearListSetCompareProc()

Specifies the procedure that a list object uses for sorting.

```
void
XrtGearListSetCompareProc(
    XrtGearObject                   list,
    XrtGearListItemCompareProc      compare_proc
    )
```

list is a list object whose compare procedure is being set; *compare_proc* is the procedure to be called.

The compare procedure must be defined to be of the following type.

```
int (*XrtGearListItemCompareProc) (
    XtPointer               item1,
    XtPointer               item2
    );
```

item1 and *item2* are the two items to be compared. The compare procedure must return the following:

- a positive integer if *item1* is greater than *item2*
- 0 if the two items are equal
- a negative integer if *item1* is less than *item2*

XrtGearListSetDestroyProc()

Specifies the procedure that the list object uses to destroy its items (if they were allocated in some special fashion).

```
void
XrtGearListSetDestroyProc(
    XrtGearObject               list,
    XrtGearListItemDestroyProc  destroy_proc
    )
```

list is a list object whose item-destroying procedure is being set; *destroy_proc* is the procedure to be called.

The item-destroying procedure must be defined to be of the following type.

```
void (*XrtGearListItemDestroyProc) (
    XrtGearObject           object,
    XtPointer               item,
    XtPointer               user_data
    );
```

object is a list object containing the item to be destroyed; *item* is the item to be destroyed; *user_data* is the user data (which is set by *XrtGearListSetUserData()*).

XrtGearListSetItem()

Sets an item in a list. Returns `False` if the index specified is invalid.

```
Boolean
XrtGearListSetItem(
    XrtGearObject           list,
    int                     index,
    XtPointer               item
    )
```

list is a list object; *index* is the location of the item to be set; *item* is a pointer to the item to be set in the list.

XrtGearListSetUserData()

Sets the user data associated with an XrtList object.

```
void
XrtGearListSetUserData(
    XrtGearObject           list,
    XtPointer               user_data
    )
```

list is a list object; *user_data* is the user data.

XrtGearListSort()

Sorts the items in a list object. If the list is already sorted, this method does nothing.

```
void
XrtGearListSort(
    XrtGearObject           list
    )
```

list is a list to be sorted.

13.5.2 XrtString Object Methods

XrtGearCvtStringToXmString()

Converts a String to an XmString. Returns the converted XmString.

```
XmString
XrtGearCvtStringToXmString(
    const String            string
    )
```

string is the String to be converted.

XrtGearCvtXmStringToString()

Converts an XmString to a String. Returns the converted String.

```
XmString
XrtGearCvtXmStringToString(
    const XmString          string
    )
```

string is the XmString to be converted.

XrtGearStringCompare()

Compares two strings using *strcmp()*.

```
int
XrtGearStringCompare(
    XrtGearObject               string1,
    XrtGearObject               string2
    )
```

string1 and *string2* are the two strings to be compared. If a string object is of type XmString, it is converted to a String for comparison.

XrtGearStringCopy()

Copies a string object and returns a pointer to the copy.

```
XrtGearObject
XrtGearStringCopy(
    XrtGearObject               string_object
    )
```

string_object is the string object to be copied.

XrtGearStringCreate()

Creates an XrtString object and returns a pointer to it.

```
XrtGearObject
XrtGearStringCreate(
    XrtGearObjectClass          class,
    XtPointer                   value
    )
```

class is the widget class, and must be either `XrtGearCharStringObjectClass` or `XrtGearXmStringObjectClass`. *value* is the text of the created string.

XrtGearStringCreateCharString()

Creates an XrtString object of class `XrtGearCharStringObjectClass` containing the specified character string, and returns a pointer to it.

```
XrtGearObject
XrtGearStringCreateCharString(
    String                  char_string
    )
```

char_string is the character string to be contained in the object. Calling this procedure is equivalent to the following:

```
XrtGearStringCreate(XrtGearCharStringObjectClass,
    (XtPointer) char_string);
```

XrtGearStringCreateIconString()

Creates an XrtString object of class `XrtGearIconStringObjectClass` containing the specified XrtGearIcon structure, and returns a pointer to it.

```
XrtGearObject
XrtGearStringCreateIconString(
    XrtGearIcon             *iconptr;
    )
```

iconptr is a pointer to the icon to be contained in the object. Calling this procedure is equivalent to the following:

```
XrtGearStringCreate(XrtGearIconStringObjectClass,
    (XtPointer) iconptr);
```

XrtGearStringCreateXmString()

Creates an XrtString object of class `XrtGearXmStringObjectClass` containing the specified XmString, and returns a pointer to it.

```
XrtGearObject
XrtGearStringCreateXmString(
    XmString                xm_string
    )
```

xm_string is the XmString to be contained in the object. Calling this procedure is equivalent to the following:

```
XrtGearStringCreate(XrtGearXmStringObjectClass,
    (XtPointer) char_string);
```

XrtGearStringDestroy()

Destroys the string object passed to it and frees all its allocated memory.

```
void
XrtGearStringDestroy(
    XrtGearObject           string_object
    )
```

string_object is the string object to be destroyed.

XrtGearStringGetValue()

Returns the value of the string object. This will be either an XmString or a String, depending on the class of the object.

```
XtPointer
XrtGearStringGetValue(
    XrtGearObject           string_object
    )
```

string_object is the string object whose value is to be retrieved.

XrtGearStringIsCharString()

Returns `True` if the specified object is of class `XrtGearCharStringObjectClass`.

```
Boolean
XrtGearStringIsCharString(
    XrtGearObject           string_object
    )
```

string_object is the object whose class is being checked.

XrtGearStringIsIconString()

Returns `True` if the specified object is of class `XrtGearIconStringObjectClass`.

```
Boolean
XrtGearStringIsIconString(
    XrtGearObject           string_object
    )
```

string_object is the object whose class is being checked.

XrtGearStringIsXmString()

Returns `True` if the specified object is of class `XrtGearXmStringObjectClass`.

```
Boolean
XrtGearStringIsXmString(
    XrtGearObject           string_object
    )
```

string_object is the object whose class is being checked.

XrtGearStringSetValue()

Sets the value of an object.

```
void
XrtGearStringSetValue(
    XrtGearObject           string_object,
    XtPointer               value
    )
```

string_object is the object whose value is to be changed; *value* is the new value. Note that *value* must be of the same type as the previous value of *string_object*.

13.5.3 XrtColorEditor Methods

XrtGearIsColorEditor()

Returns `True` if the object exists and is an *XrtGearColorEditor* object.

```
Boolean
XrtGearIsColorEditor(
    XrtGearObject   editor
    )
```

XrtGearColorEditorDestroy()

Destroys the *XrtGearColorEditor* and frees any allocated memory. If the editor was visible, it is popped down before being destroyed.

```
void
XrtGearColorEditorDestroy(
    XrtGearObject   editor
    )
```

XrtGearColorEditorCreate()

Creates an *XrtGearColorEditor* in a popup shell with the specified widget as a parent and with the specified name if popup is `True`. If popup is `False`, then the color editor is created as a direct child of the provided parent. To set resource such as XmForm attachments on the color editor, get the child `XRTGEAR_COLOR_EDITOR_TOP_WIDGET` using the method *XrtColorEditorGetChild()* and set the resources on that widget.

```
XrtGearObject
XrtGearColorEditorCreate(
    Widget          parent,
    String          name,
    Boolean         popup
    )
```

parent is the parent of the color editor; *name* is the name of the color editor; *popup* determines how the color editor is created.

XrtGearColorEditorGetName()

Returns the name of the specified XRT/gear color editor. The name should not be freed after use.

```
String
XrtGearColorEditorGetName(
    XrtGearObject   editor
    )
```

XrtGearColorEditorShow()

Pops up the color editor shell.

```
Boolean
XrtGearColorEditorShow(
    XrtGearObject   editor,
    Position        x,
    Position        y
    )
```

XrtGearColorEditorHide()

Pops down the color editor shell if it was created as a popup. The current state is preserved and the editor is not destroyed. Returns `True` if the passed object is a color editor, is currently showing and was successfully hidden.

```
Boolean
XrtGearColorEditorHide(
    XrtGearObject   editor
    )
```

XrtGearColorEditorIsShowing()

Returns `True` if the color editor shell was created as a popup and is currently popped up. If it's not a popup, the returned value is undefined.

```
Boolean
XrtGearColorEditorIsShowing(
    XrtGearObject   editor
    )
```

XrtGearColorEditorGetChild()

Returns the specified child of the color editor shell. This child should not be destroyed. This function is intended to allow modification of the specified widget's properties for which convenience functions are not provided.

```
Widget
XrtGearColorEditorGetChild(
    XrtGearObject       editor,
    XrtColorEditorChild child
    )
```

child is one of the *XrtColorEditorChild* enumeration values.

XrtGearColorEditorSetColorName()

Sets the color currently selected in the color editor by its name.

```
void
XrtGearColorEditorSetColorName(
    XrtGearObject   editor,
    String          color
    )
```

color is either the actual text name of the color (i.e. "Misty Rose") or the Hex color string (i.e. #fff).

Note that though the above are the only two formats supported, almost all strings supported by XParseColor can be used (i.e. "TekHVC:0.0/100.0/0.0"). This string is converted to #RRGGBB format when displayed.

XrtGearColorEditorSetColorList()

Replaces the default color picklist used by the combo box with a user supplied list. This list must be a `NULL` terminated array of strings. If `NULL` is passed as the color list, the default list is restored. These color names are looked up in the *rgb.txt* color database when they need to be displayed. The most common location for this file is */usr/lib/X11/rgb.txt*.

```
void
XrtGearColorEditorSetColorList(
    XrtGearObject   editor,
    String          *color_list
    )
```

XrtGearColorEditorSetRGBLabels()

Sets the labels used for the Red, Green and Blue sliders. If `NULL` is passed for any string, the current label is not changed. The strings are copied and can be freed by the user after this function call.

```
void
XrtGearColorEditorSetRGBLabels(
    XrtGearObject   editor,
    XmString        red,
    XmString        green,
    XmString        blue
    )
```

XrtGearColorEditorSetRGBValues()

Updates the RGB sliders with the provided values. Must be in the range 0..255. The color text widget and slider text widgets are also updated.

```
void
XrtGearColorEditorSetRGBValues(
    XrtGearObject   editor,
    unsigned int    red,
    unsigned int    green,
    unsigned int    blue
    )
```

XrtGearColorEditorSetHSBLabels()

Sets the labels used for the Hue, Saturation and Brightness sliders. If `NULL` is passed for any string, the current label is not changed. The strings are copied and can be freed by the user after this function call.

```
void
XrtGearColorEditorSetHSBLabels(
    XrtGearObject   editor,
    XmString        hue,
    XmString        saturation,
    XmString        brightness
    )
```

XrtGearColorEditorSetHSBValues()

Updates the HSB sliders with the provided values. Must be in the range: Saturation/Brightness 0..100, Hue 0..360. The HSB values are converted to RGB and the RGB values are updated.

```
void
XrtGearColorEditorSetHSBValues(
    XrtGearObject   editor,
    unsigned int    hue,
    unsigned int    saturation,
    unsigned int    brightness
    )
```

XrtGearColorEditorGetColorName()

Returns the `#RRGGBB` string that represents the currently selected color. Memory is allocated for this string and it is the caller's responsibility to free this memory. If match is `True`, the current color list is searched for a matching entry and if found, its name is returned. This can involve many calls to the X server depending on the size of the list and can impact an application's performance.

```
String
XrtGearColorEditorGetColorName(
    XrtGearObject   editor,
    Boolean         match
    )
```

XrtGearColorEditorGetRGBLabels()

Returns the XmStrings currently used to label the red, green and blue sliders. These values are not copied and should not be freed after this function call.

```
void
XrtGearColorEditorGetRGBLabels(
    XrtGearObject   editor,
    XmString        *red,
    XmString        *green,
    XmString        *blue
    )
```

XrtGearColorEditorGetRGBValues()

Returns the red, green and blue values of the current color in the range 0..255.

```
void
XrtGearColorEditorGetRGBValues(
    XrtGearObject   editor,
    unsigned int    *red,
    unsigned int    *green,
    unsigned int    *blue
    )
```

XrtGearColorEditorGetHSBLabels()

Returns the XmStrings currently used to label the hue, saturation and brightness sliders. These values are not copied and should not be freed after this function call.

```
void
XrtGearColorEditorGetHSBLabels(
    XrtGearObject   editor,
    XmString        *hue,
    XmString        *saturation,
    XmString        *brightness
    )
```

XrtGearColorEditorGetHSBValues()

Returns the HSB values of the current color.

```
void
XrtGearColorEditorGetHSBValues(
    XrtGearObject   editor,
    unsigned int    *hue,
    unsigned int    *saturation,
    unsigned int    *brightness
    )
```

XrtGearColorEditorSetEditTypes()

Sets the type of color editing allowed. The value passed in is an OR of type:

```
XRTGEAR_COLOR_EDITOR_BYNAME
XRTGEAR_COLOR_EDITOR_BYRGB
XRTGEAR_COLOR_EDITOR_BYHSB
```

The appropriate widgets (including toggle buttons) are managed and unmanaged by this function. "`types" = = 0` is not allowed.

```
Boolean
XrtGearColorEditorSetEditTypes(
    XrtGearObject          editor,
    XrtColorEditorEditTypes types
    )
```

XrtGearColorEditorGetEditTypes()

Returns the types of color editing allowed. This value is an OR of type:

```
XRTGEAR_COLOR_EDITOR_BYNAME
XRTGEAR_COLOR_EDITOR_BYRGB
XRTGEAR_COLOR_EDITOR_BYHSB
```

```
XrtColorEditorEditTypes
XrtGearColorEditorGetEditTypes(
    XrtGearObject   editor
    )
```

XrtGearColorEditorAddOKCallback()

Registers the provided callback procedure and client data against the OK button of the color editor dialog.

```
void
XrtGearColorEditorAddOKCallback(
    XrtGearObject   editor,
    XtCallbackProc  callback,
    XtPointer       client_data
    )
```

XrtGearColorEditorAddCancelCallback()

Registers the provided callback procedure and client data against the Cancel button of the color editor dialog.

```
void
XrtGearColorEditorAddCancelCallback(
    XrtGearObject   editor,
    XtCallbackProc  callback,
    XtPointer       client_data
    )
```

XrtGearColorEditorSetCloseColorProc()

Sets the function used to find a color close to the requested color when the colormap is full. The default procedure is *XrtGearSampleColorFunction* in the file *color_func.c*. This file is provided in *$XRTHOME/src/gear/utils*. The documentation is also in that file.

```
void
XrtGearColorEditorSetCloseColorProc(
    XrtGearObject           editor,
    XrtColorEditorCloseProc proc
    )
```

XrtGearColorEditorAllocateColor()

Allocates a new color cell with the current values and returns it. The cell is allocated read/write or read-only, depending on the value of *read_only*. It is the caller's responsibility to free this color with *XFreeColors()* when it is no longer required.

```
Pixel
XrtGearColorEditorAllocateColor(
    XrtGearObject   editor,
    Boolean         read_only
    )
```

13.5.4 Icon Manipulation Methods

XrtGearIconGetSize()

Returns the width and height of the specified icon.

```
Boolean
XrtGearIconGetSize(
    Display         *display,
    XrtGearIcon     *icon,
    Dimension       *width
    Dimension       *height,
    )
```

display is the Display object containing information about the display on which the icon appears; *icon* is the icon being measured; *width* is the returned width; *height* is the returned height. If either the width or the height is not needed, *width* or *height* can be passed `NULL`.

XrtGearIconLoadFromXpmData()

Loads icon image data using the pixmaps created from the specified XPM data structure. If the file has been previously loaded, it is pulled from a cache. Returns `False` on error, `True` on success.

```
Boolean
XrtGearIconLoadFromXpmData
    Widget          w,
    String          *data,
    String          name,
    XrtGearIcon     *icon
    )
```

w is the widget for which the icon image data is to be loaded; *data* is the XPM data structure from which the pixmaps are created; *icon* is the icon being filled in; *name* is the icon's name.

XrtGearIconLoadFromXpmFile()

Loads icon image data using the pixmaps created from the specified file. If the file has been previously loaded, it is pulled from a cache. Returns `False` on error, `True` on success.

```
Boolean
XrtGearIconLoadFromXpmFile(
    Widget          w,
    String          filename,
    XrtGearIcon     *icon
    )
```

w is the widget for which the icon image data is to be loaded; *filename* is the name of the file from which the pixmaps are created; *icon* is the icon being filled in.

Part II

Reference Appendices

A

Data Types

This appendix lists the XRT/gear data types in alphabetical order. The C language definition of structures is also provided.

XrtGearAction

Enumeration used by Tab Manager with the `XmNxrtGearPageCallback` resource (the *action* member passed to each callback):

```
XRTGEAR_ACTION_OTHERS
XRTGEAR_ACTION_SCROLL_INCR
XRTGEAR_ACTION_SCROLL_DECR
XRTGEAR_ACTION_SET_VALUES
XRTGEAR_ACTION_TAB
```

XrtGearAlignerAlignment

Enumeration used by Aligner with the `XmNxrtGearAlignerAlignment` resource to specify label position:

```
XRTGEAR_ALIGNER_BEGINNING
XRTGEAR_ALIGNER_CENTER
XRTGEAR_ALIGNER_END
```

XrtGearAlignment

Enumeration used by the Column object with the `XmNxrtGearLabelAlignment` and `XmNxrtGearNodeAlignment` resources to specify alignment of text within a label or column:

```
XRTGEAR_ALIGN_BEGINNING
XRTGEAR_ALIGN_CENTER
XRTGEAR_ALIGN_END
```

XrtGearAnchorSide

Enumeration used by Tab Button with the `XmNxrtGearAnchorSide` resource to specify the side of the widget to be joined to the parent:

```
XRTGEAR_ANCHOR_SIDE_BOTTOM
XRTGEAR_ANCHOR_SIDE_LEFT
XRTGEAR_ANCHOR_SIDE_RIGHT
XRTGEAR_ANCHOR_SIDE_TOP
```

XrtGearArrowType

Enumeration used by ComboBox with the `XmNxrtGearArrowType` resource to specify the type of arrow displayed:

```
XRTGEAR_ARROW_CUSTOM
XRTGEAR_ARROW_NONE
XRTGEAR_ARROW_WEDGE
XRTGEAR_ARROW_WITH_UNDERBAR
```

XrtGearAttachPolicy

Enumeration used by Tab Manager with the `XmNxrtGearAttachPolicy` resource to specify whether automatic attachment of Tab Buttons to pages is to be performed when required:

```
XRTGEAR_ATTACH_ALWAYS
XRTGEAR_ATTACH_NONE
```

XrtGearBorderLocation

Enumeration used by Outliner with the `XmNxrtGearBorderLocation` resource to specify where the border is drawn:

```
XRTGEAR_BORDER_LOC_LABELS
XRTGEAR_BORDER_LOC_NODES
XRTGEAR_BORDER_LOC_LABELS_SEPARATED_FROM_NODES
XRTGEAR_BORDER_LOC_LABELS_AND_NODES
XRTGEAR_BORDER_LOC_EACH_LABEL_SEPARATED_FROM_NODES,
XRTGEAR_BORDER_LOC_LABELS_AND_NODES
```

XrtGearBorderType

Enumeration used by Enhanced Label and Outliner with the `XmNxrtGearBorderType` resource and by Widget Tips with the `XmNxrtGearTipsBorderType` resource to specify the style used for the label border:

```
XRTGEAR_BORDER_NONE
XRTGEAR_BORDER_BEVEL
XRTGEAR_BORDER_ETCHED_IN
XRTGEAR_BORDER_ETCHED_OUT
XRTGEAR_BORDER_FRAME_IN
XRTGEAR_BORDER_FRAME_OUT
XRTGEAR_BORDER_IN
XRTGEAR_BORDER_OUT
XRTGEAR_BORDER_SHADOW
XRTGEAR_BORDER_PLAIN
```

XrtGearChildType

Enumeration used by Tab Manager with the `XmNxrtGearChildType` resource to specify the child type:

```
XRTGEAR_CHILD_TYPE_INTERNAL
XRTGEAR_CHILD_TYPE_PAGE
XRTGEAR_CHILD_TYPE_TAB
```

XrtColorEditorChildType

Enumeration used by Color Editor to specify the child type:

```
XRTGEAR_COLOR_EDITOR_TEXT_TOGGLE
XRTGEAR_COLOR_EDITOR_RGB_TOGGLE
XRTGEAR_COLOR_EDITOR_HSB_TOGGLE
XRTGEAR_COLOR_EDITOR_TOP_WIDGET
XRTGEAR_COLOR_EDITOR_COLOR_COMBO
XRTGEAR_COLOR_EDITOR_COLOR_LABEL
XRTGEAR_COLOR_EDITOR_COLOR_TEXT
XRTGEAR_COLOR_EDITOR_COLOR_PREVIEW
XRTGEAR_COLOR_EDITOR_OK_BUTTON
XRTGEAR_COLOR_EDITOR_CANCEL_BUTTON
```

XrtGearColumnMoveCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearColumnMoveCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          column;             /* read-only */
    Widget          target_sibling;     /* read-only */
    Boolean         doit;
} XrtGearColumnMoveCallbackStruct;
```

XrtGearColumnResizePolicy

Enumeration used by Outliner with the XrtColumn object's `XmNxrtGearWidthResizePolicy` resource to specify the column resizing policy:

```
XRTGEAR_COLUMN_RESIZE_ALL
XRTGEAR_COLUMN_RESIZE_LABELS_ONLY
XRTGEAR_COLUMN_RESIZE_NONE
```

XrtGearColumnSelectionCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearColumnSelectionCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          old_column;         /* read-only */
    Widget          new_column;
    Boolean         doit;
} XrtGearColumnSelectionCallbackStruct;
```

XrtGearCursorTrackingCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearCursorTrackingCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          object;             /* read-only */
    Cursor          old_cursor;         /* read-only */
    Cursor          new_cursor;
} XrtGearCursorTrackingCallbackStruct;
```

XrtGearDirection

Enumeration used by Outliner to specify the search direction when calling *XrtGearNodeSearchTree()*:

```
XRTGEAR_DIRECTION_FORWARD
XRTGEAR_DIRECTION_BACKWARD
```

XrtGearDisplayCallbackStruct

Structure used by Widget Tips to define the information passed to callbacks on the `XmNxrtGearDisplayCallback` list:

```
typedef struct {
    int             reason;                 /* read-only */
    XEvent          *event;                 /* read-only */
    Widget          widget;                 /* read-only */
    Boolean         doit;
    XmString        text;
} XrtGearDisplayCallbackStruct;
```

XrtGearDisplayPolicy

Enumeration used by Outliner with the `XmNxrtGearScrollBarHorizDisplayPolicy` and `XmNxrtGearScrollBarVertDisplayPolicy` resources to specify scroll bar display behavior, and by Tab Manager with the `XmNxrtGearArrowDisplayPolicy` resource to specify paging arrow display behavior:

```
XRTGEAR_DISPLAY_AS_NEEDED
XRTGEAR_DISPLAY_ALWAYS
XRTGEAR_DISPLAY_NEVER
```

XrtGearEditType

Enumeration used by Outliner to determine how a cell is edited:

```
XRTGEAR_EDIT_TYPE_WINDOWS/* the default, click to select, click
                            again to edit. Text in the editor is
                            initially selected. */
XRTGEAR_EDIT_TYPE_MOTIF  /* click to select and edit */
```

XrtGearEnhancedSizeCallbackStruct

Structure used by the Column object to define the information passed to callbacks on the `XmNxrtGearResizeCallback` list:

```
typedef struct {
    int             reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Dimension       width;
    Dimension       height;
    Dimension       old_width;          /* read-only */
    Dimension       old_height;         /* read-only */
    Boolean         doit;
} XrtGearEnhancedSizeCallbackStruct;
```

XrtGearFolderState

Enumeration used by the FolderNode object with the `XmNxrtGearFolderState` resource to specify the folder state:

```
XRTGEAR_FOLDERSTATE_CLOSED
XRTGEAR_FOLDERSTATE_OPEN_FOLDERS
XRTGEAR_FOLDERSTATE_OPEN_ITEMS
XRTGEAR_FOLDERSTATE_OPEN_ALL
```

XrtGearIcon

Structure used by the NodeStyle object and the Progress widget to represent an icon:

```
typedef struct {
    Pixmap          icon;
    Pixmap          icon_mask;
} XrtGearIcon;
```

XrtGeaImportPolicy

Enumeration used by the Outliner widget's `XmNxrtGearNodeImportPolicy` resource to specify drag and drop importing restrictions:

```
XRTGEAR_IMPORT_NOTHING
XRTGEAR_IMPORT_ANYTHING
XRTGEAR_IMPORT_WITHIN_APP
XRTGEAR_IMPORT_WITHIN_WIDGET
```

XrtGearIndentPolicy

Enumeration used by Outliner with the `XmNxrtGearNodeIndentPolicy` resource to determine the amount of indenting to perform:

```
XRTGEAR_INDENT_PIXEL
XRTGEAR_INDENT_CHAR
XRTGEAR_INDENT_NONE
```

XrtGearIndicator

Enumeration used by Enhanced Toggle with the `XmNxrtGearIndicator` resource to specify the indicator style:

```
XRTGEAR_INDICATOR_NONE
XRTGEAR_INDICATOR_CHECK
XRTGEAR_INDICATOR_CHECK_BOX
XRTGEAR_INDICATOR_CIRCLE
XRTGEAR_INDICATOR_CIRCLE_BOX
XRTGEAR_INDICATOR_CROSS
XRTGEAR_INDICATOR_CROSS_BOX
XRTGEAR_INDICATOR_FILL
XRTGEAR_INDICATOR_PIXMAP
```

XrtGearLineStyle

Enumeration used by the NodeStyle object with the `XmNxrtGearLineStyle` resource to specify the line style:

```
XRTGEAR_LINESTYLE_NONE
XRTGEAR_LINESTYLE_DOTTED
XRTGEAR_LINESTYLE_SOLID
```

XrtGearListChangedCallbackStruct

Structure used by the List object's *XrtGearListSetChangedProc()* method to specify the procedure to be called whenever a list object is changed.

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XtPointer       user_data;
    XtPointer       modified_item;
    int             position;
} XrtGearListChangedCallbackStruct;
```

XrtGearLocation

Enumeration used by the Node object's *XrtGearNodeFromXEvent()* and *XrtGearNodeFromXY()* methods to specify where an event has occurred relative to a node:

```
XRTGEAR_LOCATION_ON_NODE
XRTGEAR_LOCATION_BEFORE_NODE
XRTGEAR_LOCATION_ON_SHORTCUT
XRTGEAR_LOCATION_ON_LABEL
XRTGEAR_LOCATION_NOWHERE
```

XrtGearMenuAlignment

Enumeration used by the ComboBox widget with the `XmNxrtGearMenuAlignment` resource to specify the alignment of the popup menu relative to the text field:

```
XRTGEAR_MENU_ALIGN_BEGINNING
XRTGEAR_MENU_ALIGN_BOTH
XRTGEAR_MENU_ALIGN_END
```

XrtGearMenuCallbackStruct

Structure used by Combo Box to define the information passed to callbacks on the `XmNxrtGearMenuCallback` list:

```
typedef struct {
    int         reason;                 /* read-only */
    XEvent      *event;                 /* read-only */
    int         position;               /* read-only */
    String      value;
    Boolean     doit;                   /* initially True */
} XrtGearMenuCallbackStruct;
```

XrtGearMovePolicy

Enumeration used by the Outliner widget's `XmNxrtGearNodeMovePolicy` resource to control drag and drop operations within the widget:

```
XRTGEAR_MOVE_NOWHERE
XRTGEAR_MOVE_WITHIN_LEVEL
XRTGEAR_MOVE_WITHIN_WIDGET
XRTGEAR_MOVE_WITHIN_APP
```

XrtGearNodeActivateCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearActivateCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          container;          /* read-only */
    Widget          node;
    int             click_count;
} XrtGearNodeActivateCallbackStruct;
```

XrtGearNodeCreateCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearNodeCreateCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          source_node;        /* read-only */
    XrtGearObject   label;
    Widget          node_style;
    XtPointer       user_data;
    Widget          new_node;
} XrtGearNodeCreateCallbackStruct;
```

XrtGearNodeCurrentChangedCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearNodeCurrentChangedCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          container;          /* read-only */
    Widget          previous_node;      /* read-only */
    Widget          new_node;
    Boolean         doit;
} XrtGearNodeCurrentChangedCallbackStruct;
```

XrtGearNodeEnterCellCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearEnterCellCallback` list:

```
typedef struct {
    XrtGearReason   reason;
    XEvent          *event;
    Widget          previous_node;
    int             previous_column;
    Widget          editor;
    Widget          new_node;       /* read/write */
    int             new_column;     /* read/write */
    Boolean         doit;           /* Initially True */
} XrtGearNodeEnterCellCallbackStruct;
```

XrtGearNodeFolderStateCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearFolderStateCallback` list:

```
typedef struct {
    XrtGearReason       reason;         /* read-only */
    XEvent              *event;         /* read-only */
    Widget              container;      /* read-only */
    Widget              node;
    XrtGearFolderState  new_state;
    XrtGearFolderState  old_state;      /* read-only */
    Boolean             doit;
} XrtGearNodeFolderStateCallbackStruct;
```

XrtGearNodeMoveCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearNodeMoveCallback` list:

```
typedef struct {
    XrtGearReason       reason;             /* read-only */
    XEvent              *event;             /* read-only */
    unsigned char       operation;          /* read-only */
    Widget              node;               /* read-only */
    Widget              old_parent;         /* read-only */
    Widget              new_parent;         /* read-only */
    Widget              new_next;
    Boolean             doit;
} XrtGearNodeMoveCallbackStruct;
```

XrtGearNodeValidateCallbackStruct

Structure used by Outliner to defind the information passed to callbacks on the `XmNxrtGearNodeValidateCallback` list:

```
typedef struct {
    XrtGearReason   reason;
    XEvent          *event;
    Widget          node;
    XrtGearObject   old_label;
    int             column;
    String          value;      /* read/write */
    Boolean         doit;       /* Initially True */
    Boolean         bell;       /* Initially True */
} XrtGearNodeValidateCallbackStruct;
```

XrtGearOutlinerImportCallbackStruct

Structure used by Widget Tips to define the information passed to callbacks on the `XmNxrtGearNodeImportCallback` list:

```
typedef struct {
    XrtGearReason       reason;             /* read-only */
    XEvent              *event;             /* read-only */
    String              target;             /* read-only */
    String              data;               /* read-only */
    unsigned long       data_length;        /* read-only */
    Widget              object;             /* read-only */
    Boolean             doit;
} XrtGearOutlinerImportCallbackStruct;
```

XrtGearPageCallbackStruct

Structure used by Tab Manager to define the information passed to callbacks on the `XmNxrtGearPageCallback` list:

```
typedef struct {
    int                 reason;             /* read-only */
    XEvent              *event;             /* read-only */
    XrtGearAction       action;             /* read-only */
    int                 prev_page_number;   /* read-only */
    Widget              prev_page_widget;   /* read-only */
    Widget              prev_tab;           /* read-only */
    Widget              tab;
    int                 page_number;
    Widget              page_widget;
    Boolean             doit;
} XrtGearPageCallbackStruct;
```

XrtGearPaneResizeCallbackStruct

Structure used by Paned Window to define the information passed to callbacks on the `XmNxrtGearPaneResizeCallback` list:

```
typedef struct {
    XrtGearReason   reason;     /* read-only */
    XEvent          *event;     /* read-only */
    Widget          child;      /* read-only */
    int             position;   /* read-only */
    Dimension       width;
    Dimension       height;
    Boolean         doit;
} XrtGearPaneResizeCallbackStruct;
```

XrtGearPrintCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearPrintCallback` list:

```
typedef struct {
    int             reason;             /* read-only */
    XEvent          *event;             /* read-only */
    int             current_page;       /* read-only */
    int             total_pages;        /* read-only */
    int             across_pages;       /* read-only */
    int             down_pages;         /* read-only */
    int             page_number;
    String          header, footer;
    FILE            *fp;
    Boolean         write_file_header;
    Boolean         write_file_trailer;
    Boolean         doit;
} XrtGearPrintCallbackStruct;
```

XrtGearProgressCallbackStruct

Structure used by the Progress widget to define the information passed to callbacks on the `XmNxrtGearProgressCallback` list:

```
typedef struct {
    int             reason;             /* read-only */
    XEvent          *event;             /* read-only */
    int             maximum;            /* read-only */
    int             minimum;            /* read-only */
    double          value_percent;      /* read-only */
    XmString        label;
    int             value;
    Boolean         continue_progression;
} XrtGearProgressCallbackStruct;
```

XrtGearProgressType

Enumeration used by the Progress widget's `XmNxrtGearProgressType` resource to specify the display type of the widget:

```
XRTGEAR_PROGRESS_BAR
XRTGEAR_PROGRESS_DISCRETE_BAR
XRTGEAR_PROGRESS_ICON
XRTGEAR_PROGRESS_STACKING_ICONS
```

XrtGearPSFontMap

Structure used by the Outliner widget's `XmNxrtGearPSFontMapList` resource to define an X font to PostScript font correspondence:

```
typedef struct {
    String          font_name, tag;
    String          PS_font;
    double          point_size;
} XrtGearPSFontMap;
```

XrtGearReason

Enumeration used by XRT/gear to specify callback reasons:

```
XRTGEAR_REASON_ACTIVATE
XRTGEAR_REASON_COLUMN_MOVE_BEGIN
XRTGEAR_REASON_COLUMN_MOVE_END
XRTGEAR_REASON_COLUMN_SELECT_BEGIN
XRTGEAR_REASON_COLUMN_SELECT_END
XRTGEAR_REASON_DISPLAY_BEGIN
XRTGEAR_REASON_DISPLAY_END
XRTGEAR_REASON_FOCUS_IN_BEGIN
XRTGEAR_REASON_FOCUS_IN_END
XRTGEAR_REASON_FOCUS_OUT
XRTGEAR_REASON_FOLDER_CHANGE_BEGIN
XRTGEAR_REASON_FOLDER_CHANGE_END
XRTGEAR_REASON_FORMAT_LABEL_BEGIN
XRTGEAR_REASON_FORMAT_LABEL_END
XRTGEAR_REASON_IMPORT_BEGIN
XRTGEAR_REASON_IMPORT_END
XRTGEAR_REASON_LIST_ADD
XRTGEAR_REASON_LIST_DELETED
XRTGEAR_REASON_LIST_MOVE
XRTGEAR_REASON_LIST_SET
XRTGEAR_REASON_MENU_BEGIN
XRTGEAR_REASON_MENU_END
XRTGEAR_REASON_NODE_CHANGE_BEGIN
XRTGEAR_REASON_NODE_CHANGE_END
XRTGEAR_REASON_NODE_CREATE_BEGIN
XRTGEAR_REASON_NODE_CREATE_END
XRTGEAR_REASON_NODE_MOVE_BEGIN
XRTGEAR_REASON_NODE_MOVE_END
XRTGEAR_REASON_PAGE_BEGIN
XRTGEAR_REASON_PAGE_END
XRTGEAR_REASON_PAGE_PRINT_BEGIN
XRTGEAR_REASON_PAGE_PRINT_END
XRTGEAR_REASON_PROGRESS
XRTGEAR_REASON_PROGRESS_BEGIN
XRTGEAR_REASON_PROGRESS_COMPLETE
XRTGEAR_REASON_RESIZE_BEGIN
XRTGEAR_REASON_RESIZE_END
XRTGEAR_REASON_SELECT_BEGIN
XRTGEAR_REASON_SELECT_END
XRTGEAR_REASON_STATE_BEGIN
XRTGEAR_REASON_STATE_END
XRTGEAR_REASON_UNSELECT_BEGIN
XRTGEAR_REASON_UNSELECT_END
```

XrtGearResizeCallbackStruct

Structure used by Tab Manager and Outliner to define the information passed to callbacks on the `XmNxrtGearResizeCallback` list:

```
typedef struct {
    int         reason;             /* read-only */
    XEvent      *event;             /* read-only */
    Dimension   width;
    Dimension   height;
} XrtGearResizeCallbackStruct;
```

XrtGearRotation

Enumeration used by Enhanced Label, Enhanced Pushbutton and Tab Button with the `XmNxrtGearPixmapRotation`, `XmNxrtGearStringRotation` and `XmNxrtGearTabRotation` resources to specify rotation:

```
XRTGEAR_ROTATE_NONE
XRTGEAR_ROTATE_90
XRTGEAR_ROTATE_180
XRTGEAR_ROTATE_270
XRTGEAR_ROTATE_UNSPECIFIED
```

XrtGearSearchPolicy

Enumeration used by Combo Box with the `XmNxrtGearSearchPolicy` resource to specify search behavior in pick lists:

```
XRTGEAR_SEARCH_SINGLE
XRTGEAR_SEARCH_MULTICHAR
```

XrtGearSelectCallbackStruct

Structure used by Outliner to define the information passed to callbacks on the `XmNxrtGearSelectionCallback` list:

```
typedef struct {
    XrtGearReason   reason;             /* read-only */
    XEvent          *event;             /* read-only */
    Widget          container;          /* read-only */
    Widget          node;
    XrtGearObject   selected_node_list; /* read-only */
    Boolean         doit;
} XrtGearSelectCallbackStruct;
```

XrtGearSelectionPolicy

Enumeration used by Outliner with the `XmNxrtGearSelectionPolicy` resource to specify restrictions on what can be selected:

```
XRTGEAR_SELECTION_POLICY_NONE
XRTGEAR_SELECTION_POLICY_SINGLE
XRTGEAR_SELECTION_POLICY_SINGLE_AUTO_SELECT
XRTGEAR_SELECTION_POLICY_RANGE
XRTGEAR_SELECTION_POLICY_MULTIPLE
XRTGEAR_SELECTION_POLICY_MULTIPLE_AUTO_UNSELECT
```

XrtGearSelectionRestrictions

Enumeration used by Outliner with the `XmNxrtGearSelectionRestrictions` resource to specify restrictions on what can be selected:

```
XRTGEAR_RESTRICT_NOTHING
XRTGEAR_RESTRICT_ITEMS
XRTGEAR_RESTRICT_FOLDERS
XRTGEAR_RESTRICT_MULTI_LEVEL
XRTGEAR_RESTRICT_ALL
```

XrtGearSizingPolicy

Enumeration used by the Outliner widget and the Column object with the `XmNxrtGearHeightSizingPolicy` and `XmNxrtGearWidthSizingPolicy` resources to specify how the preferred height or width of the displayed portion of the tree is calculated:

```
XRTGEAR_SIZE_PIXEL
XRTGEAR_SIZE_CHAR
XRTGEAR_SIZE_COLUMN
XRTGEAR_SIZE_NODE
XRTGEAR_SIZE_LABEL_VARIABLE
XRTGEAR_SIZE_VARIABLE
```

XrtGearSnapArg

Structure used by the Widget Snapshot utility to pass attribute-value pairs to *XrtGearSnapPrintPixmap()*:

```
typedef struct {
    XrtGearSnapDrawArg  name;
    XtArgVal            value;
} XrtGearSnapArg;
```

XrtGearSnapDrawArg

Enumeration used by the Widget Snapshot utility, listing the arguments and values that can be passed to *XrtGearSnapPrintPixmap()*:

```
XRTGEAR_SNAP_ALL
XRTGEAR_SNAP_AS_IS
XRTGEAR_SNAP_AUTO
XRTGEAR_SNAP_BEST
XRTGEAR_SNAP_CM
XRTGEAR_SNAP_COLOR
XRTGEAR_SNAP_INCHES
XRTGEAR_SNAP_LANDSCAPE
XRTGEAR_SNAP_MARGIN_BOTTOM
XRTGEAR_SNAP_MARGIN_LEFT
XRTGEAR_SNAP_MARGIN_RIGHT
XRTGEAR_SNAP_MARGIN_TOP
XRTGEAR_SNAP_MONO
XRTGEAR_SNAP_NONE
XRTGEAR_SNAP_ORIENTATION
XRTGEAR_SNAP_PAPERSIZE_HEIGHT
XRTGEAR_SNAP_PAPERSIZE_WIDTH
XRTGEAR_SNAP_PORTRAIT
XRTGEAR_SNAP_SCALE
XRTGEAR_SNAP_SHOWPAGE
XRTGEAR_SNAP_UNITS
XRTGEAR_SNAP_NUM_COPIES
```

XrtGearState

Enumeration used by Enhanced Toggle with the `XmNxrtGearState` resource to specify the state of the button (defined as a convenience):

```
XRTGEAR_STATE_OFF
XRTGEAR_STATE_ON
XRTGEAR_STATE_INDETERMINATE
```

XrtGearStateChangedCallbackStruct

Structure used by Enhanced Toggle to define the information passed to callbacks on the `XmNxrtGearStateChangedCallback` list:

```
typedef struct {
    int                reason;     /* read-only */
    XEvent             *event;     /* read-only */
    int                state;
} XrtGearStateChangedCallbackStruct;
```

XrtGearStringLayout

Enumeration used by Enhanced Label, Enhanced Pushbutton and Tab Button with the `XmNxrtGearStringLayout` resource to specify the relative position of the text with respect to the pixmap:

```
XRTGEAR_STRING_BOTTOM
XRTGEAR_STRING_CENTER
XRTGEAR_STRING_LEFT
XRTGEAR_STRING_RIGHT
XRTGEAR_STRING_TOP
```

XrtGearStyle

Enumeration used by Tab Button with the `XmNxrtGearStyle` resource to specify the tab button style:

```
XRTGEAR_STYLE_CHAMFERED
XRTGEAR_STYLE_ROUNDED
XRTGEAR_STYLE_SLANTED
XRTGEAR_STYLE_SQUARE
```

XrtGearTabSide

Enumeration used by Tab Manager with the `XmNxrtGearTabSide` resource to indicate the side on which tabs are displayed:

```
XRTGEAR_SIDE_BOTTOM
XRTGEAR_SIDE_LEFT
XRTGEAR_SIDE_RIGHT
XRTGEAR_SIDE_TOP
```

XrtGearTextClipPolicy

Enumeration used by Outliner with the `XmNxrtGearTextHorizClipPolicy` resource to specify how to handle text which is too large to fit:

```
XRTGEAR_CLIP_TRUNCATE
XRTGEAR_CLIP_TRUNCATE_WITH_ARROW
XRTGEAR_CLIP_COMPRESS_WITH_ELLIPSIS
```

XrtGearType

Enumeration used by Enhanced Label, Enhanced Pushbutton and Tab Button with the `XmNxrtGearLabelType` resource to specify label type:

```
XRTGEAR_TYPE_UNSPECIFIED
XRTGEAR_TYPE_PIXMAP
XRTGEAR_TYPE_STRING
XRTGEAR_TYPE_STRING_PIXMAP
```

XrtGearWidgetResizePolicy

Enumeration used by Outliner with the `XmNxrtGearHeightResizePolicy` and `XmNxrtGearWidthResizePolicy` resources to specify when a widget should try to resize itself:

```
XRTGEAR_WIDGET_RESIZE_VARIABLE
XRTGEAR_WIDGET_RESIZE_GROW_ONLY
XRTGEAR_WIDGET_RESIZE_FIXED
```

B

Resource Types and Classes

This appendix lists the resource type and resource class for each XRT/gear resource.

B.1 Aligner

Name	Class	Type
XmNxrtGearColumnSpacing	XmCXrtGearColumnSpacing	XmRHorizontalDimension
XmNxrtGearMarginHeight	XmCXrtGearMarginHeight	XmRHorizontalDimension
XmNxrtGearMarginWidth	XmCXrtGearMarginWidth	XmRHorizontalDimension
XmNxrtGearOrientation	XmCXrtGearOrientation	XmROrientation
XmNxrtGearRowSpacing	XmCXrtGearRowSpacing	XmRHorizontalDimension

B.1.1 Aligner Constraint Resources

Name	Class	Type
XmNxrtGearAlignerAlignment	XmCXrtGearAlignerAlignment	XmRXrtGearAlignerAlignment
XmNxrtGearAlignBaseline	XmCXrtGearAlignBaseline	XmRBoolean
XmNxrtGearPosition	XmCXrtGearPosition	XmRInt
XmNxrtGearResizeHeight	XmCXrtGearResizeHeight	XmRBoolean
XmNxrtGearResizeWidth	XmCXrtGearResizeWidth	XmRBoolean

B.2 Combo Box

Name	Class	Type
XmNxrtGearAlignment	XmCXrtGearAlignment	XmRXrtGearMenuAlignment
XmNxrtGearArrow	XmCReadOnly	XmRWidget
XmNxrtGearArrowClass	XmCReadOnly	XmRPointer
XmNxrtGearArrowType	XmCXrtGearArrowType	XmRxrtGearArrowType
XmNxrtGearAutoComplete	XmCBoolean	XmRBoolean
XmNxrtGearMenuCallback	XmCCallback	XmRCallback
XmNxrtGearMenuList	XmCReadOnly	XmRWidget
XmNxrtGearPickList	XmCXrtGearPickList	XmRStringTable
XmNxrtGearPickListIndex	XmCXrtGearPickListIndex	XmRInt
XmNxrtGearSearchPolicy	XmCXrtGearSearchPolicy	XmRXrtGearSearchPolicy
XmNxrtGearSpacing	XmCXrtGearSpacing	XmRHorizontalDimension
XmNxrtGearTextField	XmCReadOnly	XmRWidget

B.3 Enhanced Label

Name	Class	Type
XmNxrtGearBorderType	XmCXrtGearBorderType	XmRXrtGearBorderType
XmNxrtGearLabelPixmap	XmCXrtGearLabelPixmap	XmRXrtGearPixmap
XmNxrtGearLabelType	XmCXrtGearLabelType	XmRXrtGearType
XmNxrtGearLayoutSpacing	XmCXrtGearLayoutSpacing	XmRDimension
XmNxrtGearMask	XmCXrtGearMask	XmRXrtGearMask
XmNxrtGearPixmapRotation	XmCXrtGearPixmapRotation	XmRXrtGearRotation
XmNxrtGearStringLayout	XmCXrtGearStringLayout	XmRXrtGearStringLayout
XmNxrtGearStringRotation	XmCXrtGearStringRotation	XmRXrtGearRotation

B.4 Enhanced Pushbutton

Name	Class	Type
XmNxrtGearActive	XmCBoolean	XmRBoolean
XmNxrtGearArmMask	XmCXrtGearArmMask	XmRXrtGearMask

Name	Class	Type
XmNxrtGearArmPixmap	XmCXrtGearArmPixmap	XmRXrtGearPixmap
XmNxrtGearLabelPixmap	XmCXrtGearLabelPixmap	XmRXrtGearPixmap
XmNxrtGearLabelType	XmCXrtGearLabelType	XmRXrtGearType
XmNxrtGearLayoutSpacing	XmCXrtGearLayoutSpacing	XmRDimension
XmNxrtGearMask	XmCXrtGearMask	XmRXrtGearMask
XmNxrtGearPixmapRotation	XmCXrtGearPixmapRotation	XmRXrtGearRotation
XmNxrtGearStringLayout	XmCXrtGearStringLayout	XmRXrtGearStringLayout
XmNxrtGearStringRotation	XmCXrtGearStringRotation	XmRXrtGearRotation

B.5 Enhanced Toggle

Name	Class	Type
XmNxrtGearIndicator	XmCXrtGearIndicator	XmRXrtGearIndicator
XmNxrtGearIndicatorMaskList	XmCXrtGearIndicatorMaskList	XmRXrtGearIndicatorMaskList
XmNxrtGearIndicatorPixmapList	XmCXrtGearIndicatorPixmapList	XmRXrtGearIndicatorPixmapList
XmNxrtGearNumStates	XmCXrtGearNumStates	XmRInt
XmNxrtGearSelectColor	XmCXrtGearSelectColor	XmRPixel
XmNxrtGearState	XmCXrtGearState	XmRXrtGearState
XmNxrtGearStateChangedCallback	XmCCallback	XmRCallback
XmNxrtGearUnselectColor	XmCXrtGearUnselectColor	XmRPixel

B.6 Outliner

Name	Class	Type
XmNxrtGearActivateCallback	XmCCallback	XmRCallback
XmNxrtGearAutoSort	XmCXrtGearAutoSort	XmRBoolean
XmNxrtGearBorderLocation	XmCXrtGearBorderLocation	XmRXrtGearBorderLocation
XmNxrtGearBorderType	XmCXrtGearBorderType	XmRXrtGearBorderType
XmNxrtGearColumnLabelsShow	XmCXrtGearColumnLabelsShow	XmRBoolean
XmNxrtGearColumnList	XmCXrtGearColumnList	XmRXrtGearColumnList
XmNxrtGearColumnMoveAllow	XmCXrtGearColumnMoveAllow	XmRBoolean

Name	Class	Type
XmNxrtGearColumnMoveCallback	XmCXrtGearColumnMoveCallback	XmRCallback
XmNxrtGearColumnOrderList	XmCXrtGearColumnList	XmRXrtGearList
XmNxrtGearColumnSelected	XmCXrtGearColumnSelected	XmRXrtGearWidget
XmNxrtGearColumnSelectionCallback	XmCXrtGearColumnSelectionCallback	XmRCallback
XmNxrtGearCursor	XmCCursor	XmRCursor
XmNxrtGearCursorResizeWidth	XmCCursor	XmRCursor
XmNxrtGearCursorTracking	XmCXrtGearCursorTracking	XmRBoolean
XmNxrtGearCursorTrackingCallback	XmCCallback	XmRCallback
XmNxrtGearDoubleBuffer	XmCXrtGearDoubleBuffer	XmRBoolean
XmNeditable	XmCBoolean	XmRBoolean
XmNxrtGearEditColumn	XmCXrtGearEditColumn	XmRInt
XmNxrtGearEditNode	XmCXrtGearEditNode	XmRWidget
XmNxrtGearEditor	XmCXrtGearEditor	XmRWidget
XmNxrtGearFolderShortcutShow	XmCBoolean	XmRBoolean
XmNxrtGearFolderStateCallback	XmCCallback	XmRCallback
XmNxrtGearFolderStateMask	XmCXrtGearFolderStateMask	XmRXrtGearFolderStateMask
XmNfontList	XmCFontList	XmRFontList
XmNxrtGearHeightPreferredCount	XmCXrtGearHeightPreferredCount	XmRInt
XmNxrtGearHeightResizePolicy	XmCXrtGearHeightResizePolicy	XmRXrtGearWidgetResizePolicy
XmNxrtGearHeightSizingPolicy	XmCXrtGearHeightSizingPolicy	XmRXrtGearSizingPolicy
XmNxrtGearHeightVirtual	XmCXrtGearHeightVirtual	XmRInt
XmNhighlightThickness	XmCHighlightThickness	XmRDimension
XmNxrtGearImportCallback	XmCCallback	XmRCallback
XmNxrtGearMarginHeight	XmCXrtGearMarginHeight	XmRDimension
XmNxrtGearMarginWidth	XmCXrtGearMarginWidth	XmRDimension
XmNxrtGearNodeChildList	XmCXrtGearNodeChildList	XmRXrtGearNodeChildList
XmNxrtGearNodeCompareProc	XmCXrtGearNodeCompareProc	XmRXrtGearNodeCompareProc
XmNxrtGearNodeCreateCallback	XmCCallback	XmRCallback
XmNxrtGearNodeCurrent	XmCXrtGearNodeCurrent	XmRXrtGearWidget
XmNxrtGearNodeCurrentChangedCallback	XmCCallback	XmRCallback
XmNxrtGearNodeEnterCellCallback	XmCCallback	XmRCallback
XmNxrtGearNodeHeight	XmCXrtGearNodeHeight	XmRDimension

Name	Class	Type
XmNxrtGearNodeImportPolicy	XmCXrtGearNodeImportPolicy	XmRXrtGearImportPolicy
XmNxrtGearNodeIndent	XmCXrtGearNodeIndent	XmRInt
XmNxrtGearNodeIndentPolicy	XmCXrtGearNodeIndentPolicy	XmRXrtGearIndentPolicy
XmNxrtGearNodeMoveCallback	XmCCallback	XmRCallback
XmNxrtGearNodeMovePolicy	XmCXrtGearNodeMovePolicy	XmRXrtGearNodeMovePolicy
XmNxrtGearNodeSpacing	XmCXrtGearNodeSpacing	XmRDimension
XmNxrtGearNodeStyleDefault	XmCXrtGearNodeStyle	XmRXrtGearNodeStyle
XmNxrtGearNodeStyleList	XmCXrtGearNodeStyleList	XmRXrtGearNodeStyleList
XmNxrtGearNodeTop	XmCXrtGearNodeTop	XmRXrtGearWidget
XmNxrtGearNodeValidateCallback	XmCCallback	XmRCallback
XmNxrtGearNodeVisibleList	XmCXrtGearNodeVisibleList	XmRXrtGearList
XmNxrtGearPrintCallback	XmCCallback	XmRCallback
XmNxrtGearPSFontMapList	XmCXrtGearPSFontMapList	XmRXrtGearPSFontMapList
XmNxrtGearRepaint	XmCXrtGearRepaint	XmRBoolean
XmNxrtGearResizeCallback	XmCCallback	XmRCallback
XmNxrtGearScrollBarHoriz	XmCXrtGearScrollBarHoriz	XmRXrtGearWidget
XmNxrtGearScrollBarHorizDisplayPolicy	XmCXrtGearScrollBarHorizDisplayPolicy	XmRXrtGearDisplayPolicy
XmNxrtGearScrollBarVert	XmCXrtGearScrollBarVert	XmRXrtGearWidget
XmNxrtGearScrollBarVertDisplayPolicy	XmCXrtGearScrollBarVertDisplayPolicy	XmRXrtGearDisplayPolicy
XmNxrtGearSelectionCallback	XmCCallback	XmRCallback
XmNxrtGearSelectedNodeList	XmCXrtGearSelectedNodeList	XmRXrtGearList
XmNxrtGearSelectionPolicy	XmCXrtGearSelectionPolicy	XmRXrtGearSelectionPolicy
XmNxrtGearSelectionRestrictions	XmCXrtGearSelectionRestrictions	XmRXrtGearSelectionRestrictions
XmNxrtGearShadowThickness	XmCXrtGearShadowThickness	XmRDimension
XmNxrtGearTextEditor	XmCXrtGearTextEditor	XmRWidget
XmNxrtGearTextHorizClipPolicy	XmCXrtGearTextHorizClipPolicy	XmRXrtGearTextClipPolicy
XmNxrtGearTreeData	XmCXrtGearTreeData	XmRXrtGearTreeData
XmNxrtGearWidthChar	XmCXrtGearWidthChar	XmRChar
XmNxrtGearWidthPreferredCount	XmCXrtGearWidthPreferredCount	XmRInt
XmNxrtGearWidthResizePolicy	XmCXrtGearWidthResizePolicy	XmRXrtGearWidgetResizePolicy
XmNxrtGearWidthSizingPolicy	XmCXrtGearWidthSizingPolicy	XmRXrtGearSizingPolicy
XmNxrtGearWidthVirtual	XmCXrtGearWidthVirtual	XmRInt

B.6.1 Column Resources

Name	Class	Type
XmNfontList	XmCFontList	XmRFontList
XmNxrtGearFontListSelected	XmCXrtGearFontListSelected	XmRFontList
XmNforeground	XmCForeground	XmRPixel
XmNheight	XmCDimension	XmRDimension
XmNxrtGearLabel	XmCXrtGearString	XmRXrtGearString
XmNxrtGearLabelAlignment	XmCXrtGearAlignment	XmRXrtGearAlignment
XmNxrtGearMarginLeft	XmCXrtGearMarginLeft	XmRDimension
XmNxrtGearMarginRight	XmCXrtGearMarginRight	XmRDimension
XmNxrtGearNodeAlignment	XmCXrtGearAlignment	XmRXrtGearAlignment
XmNxrtGearResizeCallback	XmCCallback	XmRCallback
XmNxrtGearSelected	XmCXrtGearSelected	XmRBoolean
XmNwidth	XmCDimension	XmRDimension
XmNxrtGearWidthMaximum	XmCXrtGearWidthMaximum	XmRDimension
XmNxrtGearWidthMinimum	XmCXrtGearWidthMinimum	XmRDimension
XmNxrtGearWidthPreferredCount	XmCXrtGearWidthPreferredCount	XmRInt
XmNxrtGearWidthResizePolicy	XmCXrtGearWidthResizePolicy	XmRXrtGearColumnResizePolicy
XmNxrtGearWidthSizingPolicy	XmCXrtGearWidthSizingPolicy	XmRXrtGearSizingPolicy
XmNx	NULL	XmRPosition

B.6.2 Node Resources

Name	Class	Type
XmNEditable	XmCBoolean	XmRBoolean
XmNxrtGearLabel	XmCXrtGearLabel	XmRXrtGearLabel
XmNxrtGearNodeStyle	XmCXrtGearNodeStyle	XmRXrtGearNodeStyle
XmNxrtGearSelected	XmCXrtGearSelected	XmRBoolean
XmNuserData	XmCUserData	XmRPointer

B.6.3 Node Folder Resources

Name	Class	Type
XmNEditable	XmCBoolean	XmRBoolean

Name	Class	Type
XmNxrtGearFolderState	XmCXrtGearFolderState	XmRXrtGearFolderState
XmNxrtGearNodeChildList	XmCXrtGearNodeChildList	XmRXrtGearNodeChildList

B.6.4 Node Style Resources

Name	Class	Type
XmNBackground	XmCBackground	XmRPixel
XmNxrtGearBackgroundSelected	XmCXrtGearBackgroundSelected	XmRPixel
XmNxrtGearFolderShortcutShow	XmCBoolean	XmRBoolean
XmNfontList	XmCFontList	XmRFontList
XmNxrtGearFontListSelected	XmCXrtGearFontListSelected	XmRFontList
XmNforeground	XmCForeground	XmRPixel
XmNxrtGearForegroundSelected	XmCXrtGearForegroundSelected	XmRPixel
XmNheight	XmCDimension	XmRDimension
XmNxrtGearIconClosed	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearIconClosedSelected	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearIconItem	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearIconItemSelected	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearIconOpen	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearIconOpenSelected	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearLayoutSpacing	XmCDimension	XmRDimension
XmNxrtGearLineColor	XmCXrtGearLineColor	XmRPixel
XmNxrtGearLineStyle	XmCXrtGearLineStyle	XmRXrtGearLineStyle
XmNxrtGearLineWidth	XmCXrtGearLineWidth	XmRDimension
XmNxrtGearShortcutShow	XmCBoolean	XmRBoolean
XmNxrtGearShortcutSize	XmCDimension	XmRDimension

B.7 Progress

Name	Class	Type
XmNxrtGearBarColor	XmCForeground	XmRPixel
XmNxrtGearBarCount	XmCXrtGearBarCount	XmRInt
XmNxrtGearBarSpacing	XmCXrtGearBarSpacing	XmRDimension
XmNxrtGearDoubleBuffer	XmCXrtGearDoubleBuffer	XmRBoolean
XmNfontList	XmCFontList	XmRFontList
XmNxrtGearIcon	XmCXrtGearIcon	XmRXrtGearIcon
XmNxrtGearLabel	XmCXrtGearLabel	XmRXrtGearLabel
XmNxrtGearLabelAlignment	XmCXrtGearLabelAlignment	XmRXrtGearAlignment
XmNxrtGearLabelFormat	XmCXrtGearLabelFormat	XmRString
XmNxrtGearLabelFormatCallback	XmCXrtGearLabelFormatCallback	XmRCallback
XmNxrtGearLabelShow	XmCXrtGearLabelShow	XmRBoolean
XmNmarginHeight	XmCMarginHeight	XmRDimension
XmNmarginWidth	XmCMarginWidth	XmRDimension
XmNmaximum	XmCXrtGearProgressMaximum	XmRInt
XmNminimum	XmCXrtGearProgressMinimum	XmRInt
XmNxrtGearProgressCallback	XmCXrtGearProgressCallback	XmRCallback
XmNxrtGearProgressTimeOut	XmCXrtGearProgressTimeOut	XmRXrtGearUlong
XmNxrtGearProgressType	XmCXrtGearProgressType	XmRXrtGearProgressType
XmNrecomputeSize	XmCRecomputeSize	XmRBoolean
XmNvalue	XmCXrtGearProgressValue	XmRInt
XmNxrtGearValuePercent	XmCXrtGearValuePercent	XmRXrtGearFloat

B.8 Tab Button

Name	Class	Type
XmNxrtGearAnchorSide	XmCXrtGearAnchorSide	XmRXrtGearAnchorSide
XmNxrtGearCornerSize	XmCXrtGearCornerSize	XmRDimension
XmNxrtGearLabelPixmap	XmCXrtGearLabelPixmap	XmRXrtGearPixmap
XmNxrtGearLabelType	XmCXrtGearLabelType	XmRXrtGearType
XmNxrtGearLayoutSpacing	XmCXrtGearLayoutSpacing	XmRDimension

Name	Class	Type
XmNxrtGearMask	XmCXrtGearMask	XmRXrtGearMask
XmNxrtGearPixmapRotation	XmCXrtGearPixmapRotation	XmRXrtGearRotation
XmNxrtGearStringLayout	XmCXrtGearStringLayout	XmRXrtGearStringLayout
XmNxrtGearStringRotation	XmCXrtGearStringRotation	XmRXrtGearRotation
XmNxrtGearStyle	XmCXrtGearStyle	XmRXrtGearStyle

B.9 Tab Manager

Name	Class	Type
XmNxrtGearActivePageNumber	XmCXrtGearActivePageNumber	XmRXrtGearActivePageNumber
XmNxrtGearActivePageWidget	XmCXrtGearActivePageWidget	XmRXrtWidget
XmNxrtGearActiveTabWidget	XmCXrtGearActiveTabWidget	XmRXrtWidget
XmNxrtGearArrowDisplayPolicy	XmCXrtGearArrowDisplayPolicy	XmRXrtGearDisplayPolicy
XmNxrtGearArrowHeight	XmCXrtGearArrowHeight	XmRDimension
XmNxrtGearArrowWidth	XmCXrtGearArrowWidth	XmRDimension
XmNxrtGearCornerSize	XmCXrtGearCornerSize	XmRDimension
XmNxrtGearFirstVisibleTab	XmCXrtGearFirstVisibleTab	XmRInt
XmNxrtGearHighlightThickness	XmCXrtGearHighlightThickness	XmRDimension
XmNxrtGearLastVisibleTab	XmCXrtGearLastVisibleTab	XmRInt
XmNxrtGearMarginHeight	XmCXrtGearMarginHeight	XmRDimension
XmNxrtGearMarginWidth	XmCXrtGearMarginWidth	XmRDimension
XmNxrtGearPageCallback	XmCCallback	XmRCallback
XmNxrtGearResizeCallback	XmCCallback	XmRCallback
XmNxrtGearShadowThickness	XmCXrtGearShadowThickness	XmRDimension
XmNxrtGearTabResize	XmCXrtGearTabResize	XmRBoolean
XmNxrtGearTabRotation	XmCXrtGearTabRotation	XmRXrtGearTabRotation
XmNxrtGearTabSide	XmCXrtGearTabSide	XmRXrtGearTabSide
XmNxrtGearTabSpacing	XmCXrtGearTabSpacing	XmRInt
XmNxrtGearTabStretch	XmCXrtGearTabStretch	XmRBoolean

B.9.1 Tab Manager Constraint Resources

Name	Class	Type
XmNxrtGearChildType	XmCXrtGearChildType	XmRXrtGearChildType
XmNxrtGearPageNumber	XmCXrtGearPageNumber	XmRInt
XmNxrtGearPageWidget	XmCXrtGearPageWidget	XmRXrtWidget
XmNxrtGearWidgetPosition	XmCXrtGearWidgetPosition	XmRInt

B.10 Toolbar

Name	Class	Type
XmNxrtGearMarginHeight	XmCXrtGearMarginHeight	XmRHorizontalDimension
XmNxrtGearMarginWidth	XmCXrtGearMarginWidth	XmRHorizontalDimension
XmNxrtGearOrientation	XmCXrtGearOrientation	XmROrientation

B.10.1 Toolbar Constraint Resources

Name	Class	Type
XmNxrtGearBottomSpace	XmCXrtGearBottomSpace	XmRHorizontalDimension
XmNxrtGearLeftSpace	XmCXrtGearLeftSpace	XmRHorizontalDimension
XmNxrtGearPosition	XmCXrtGearPosition	XmRInt
XmNxrtGearRightSpace	XmCXrtGearRightSpace	XmRHorizontalDimension
XmNxrtGearTopSpace	XmCXrtGearTopSpace	XmRHorizontalDimension

B.11 Widget Snapshot

Name	Class	Type
XmNxrtGearSnapNoShapeClip	XmCXrtGearSnapNoShapeClip	XmRBoolean

B.12 Widget Tips

Name	Class	Type
XmNxrtGearAlignment	XmCXrtGearAlignment	XmRAlignment
XmNxrtGearBackground	XmCXrtGearBackground	XmRPixel
XmNxrtGearDelay	XmCXrtGearDelay	XmRInt
XmNxrtGearDisplayCallback	XmCXrtGearDisplayCallback	XmRPointer
XmNxrtGearFontList	XmCXrtGearFontList	XmRFontList
XmNxrtGearForeground	XmCXrtGearForeground	XmRPixel

Name	Class	Type
XmNxrtGearGroup	XmCXrtGearGroup	XmRXrtGearGroup
XmNxrtGearShadowThickness	XmCXrtGearShadowThickness	XmRDimension
XmNxrtGearText	XmCXrtGearText	XmRXmString
XmNxrtGearTipsBorderType	XmCXrtGearTipsBorderType	XmRXrtGearBorderType

B.13 Paned Window

Name	Class	Type
XmNxrtGearMarginHeight	XmCXrtGearMarginHeight	XmRDimension
XmNxrtGearMarginWidth	XmCXrtGearMarginWidth	XmRDimension
XmNxrtGearOrientation	XmCOrientation	XmROrientation
XmNxrtGearPaneResizeCallback	XmCXrtGearPaneResizeCallback	XmRCallback
XmNxrtGearRecomputeSize	XmCXrtGearRecomputeSize	XmRBoolean
XmNxrtGearSashHeight	XmCXrtGearSashHeight	XmRDimension
XmNxrtGearSashIndent	XmCXrtGearSashIndent	XmRPosition
XmNxrtGearSashShadowThickness	XmCXrtGearSashShadowThickness	XmRDimension
XmNxrtGearSashWidth	XmCXrtGearSashWidth	XmRDimension
XmNxrtGearSeparatorShow	XmCBoolean	XmRBoolean
XmNxrtGearSpacing	XmCXrtGearSpacing	XmRDimension

B.13.1 Paned Window Constraint Resources

Name	Class	Type
XmNxrtGearAllowResize	XmCBoolean	XmRBoolean
XmNxrtGearMinimum	XmCXrtGearMinimum	XmRDimension
XmNxrtGearMaximum	XmCXrtGearMaximum	XmRDimension
XmNxrtGearPosition	XmCXrtGearPosition	XmRInt
XmNxrtGearSkipAdjust	XmCBoolean	XmRBoolean

C

Resource File and GUI Builder Usage

This appendix lists the syntax required for resource strings in resource files so that they are recognized by the XRT/gear resource converters. It also lists a sample resource file. XRT/gear only defines resource converters for the data types not already defined by Xt and Motif.

All XRT/gear resources can be set in a resource file except resources that use a procedure to set their value.

C.1 Syntax

Comments
Any text after an exclamation mark (`!`) on a line is ignored.

Continuation
Lines are continued on subsequent lines in the file by making a backslash (`\`) the last character in the line.

Booleans
"True" or "False" in upper, lower, or mixed case.

Enumerated Types
All resources with enumerated types use a consistent naming convention based on the enumerated values. For all resource types, the leading `XRTGEAR_` is dropped. The resource values can be entered in upper, lower, or mixed case. For example, each of the following resource specifications causes the value `XRTGEAR_STYLE_SQUARE` to be used for `XmNxrtGearStyle` in a Tab Button widget:

```
*.XmXrtTabButton.xrtGearStyle: STYLE_SQUARE
*.XmXrtTabButton.xrtGearStyle: style_square
*.XmXrtTabButton.xrtGearStyle: StyLe_SquArE
```

D

UIL Arguments

XRT/gear enables you to use OSF's User Interface Language (UIL) to create widgets. Since UIL only supports data types used by standard Motif widgets, some XRT/gear resources cannot be set using UIL, but can be set in an application with an `MrmNcreateCallback`.

This appendix provides an alphabetical listing of the UIL arguments, data types and reasons for each XRT/gear widget.

Example programs that use UIL are located in *$XRTHOME/src/gear/examples/simple_cpp.c* (using *simple_cpp.uil*) and *$XRTHOME/src/gear/examples/complex_cpp.c* (using *complex_cpp.uil*).

D.1 Aligner Resources

UIL Argument Name	Argument Type
XmNxrtGearAlignBaseline	boolean
XmNxrtGearAlignerAlignment	integer
XmNxrtGearColumnSpacing	integer
XmNxrtGearMarginHeight	integer
XmNxrtGearMarginWidth	integer
XmNxrtGearOrientation	integer
XmNxrtGearResizeHeight	boolean
XmNxrtGearResizeWidth	boolean
XmNxrtGearRowSpacing	integer

D.2 Combo Box Resources

UIL Argument Name	Argument Type
XmNxrtGearAlignment	integer
XmNxrtGearArrow[a]	widget_ref
XmNxrtGearArrowType	integer
XmNxrtGearAutoComplete	Boolean
XmNxrtGearMenuList[a]	widget_ref
XmNxrtGearPickListIndex	integer
XmNxrtGearSearchPolicy	integer
XmNxrtGearTextField[a]	widget_ref

a. Get only.

Reasons
XmNxrtGearMenuCallback

D.3 Enhanced Label Resources

UIL Argument Name	Argument Type
XmNxrtGearBorderType	integer
XmNxrtGearLabelPixmap	icon
XmNxrtGearLabelType	integer
XmNxrtGearLayoutSpacing	integer
XmNxrtGearMask	pixmap
XmNxrtGearPixmapRotation	integer
XmNxrtGearStringLayout	integer
XmNxrtGearStringRotation	integer

D.4 Enhanced Pushbutton Resources

UIL Argument Name	Argument Type
XmNxrtGearActive	Boolean
XmNxrtGearArmMask	pixmap
XmNxrtGearLabelPixmap	icon
XmNxrtGearLabelType	integer
XmNxrtGearLayoutSpacing	integer
XmNxrtGearMask	pixmap
XmNxrtGearPixmapRotation	integer
XmNxrtGearStringLayout	integer
XmNxrtGearStringRotation	integer

D.5 Enhanced Toggle Resources

UIL Argument Name	Argument Type
XmNxrtGearIndicator	integer
XmNxrtGearNumStates	integer
XmNxrtGearSelectColor	color
XmNxrtGearState	integer
XmNxrtGearUnselectColor	color

Reasons
XmNxrtGearStateChangedCallback

D.6 Outliner Resources

UIL Argument Name	Argument Type
XmNxrtGearAutoSort	boolean
XmNxrtGearBorderLocation	integer
XmNxrtGearColumnLabelsShow	boolean
XmNxrtGearColumnMoveAllow	boolean
XmNxrtGearColumnSelected	widget_ref

UIL Argument Name	Argument Type
XmNxrtGearCursorTracking	boolean
XmNxrtGearDoubleBuffer	boolean
XmNxrtGearEditor	widget
XmNxrtGearEditNode	widget
XmNxrtGearEditColumn	integer
XmNxrtGearFolderStateMask	integer
XmNxrtGearHeightPreferredCount	integer
XmNxrtGearHeightResizePolicy	integer
XmNxrtGearHeightSizingPolicy	integer
XmNxrtGearHeightVirtual[a]	integer
XmNxrtGearHighlightOnDrag	boolean
XmNxrtGearNodeCheckForDuplicates	boolean
XmNxrtGearNodeCurrent	widget_ref
XmNxrtGearNodeHeight	integer
XmNxrtGearNodeImportPolicy	integer
XmNxrtGearNodeIndent	integer
XmNxrtGearNodeMovePolicy	integer
XmNxrtGearNodeSpacing	integer
XmNxrtGearNodeStyleDefault	widget_ref
XmNxrtGearNodeTop	widget_ref
XmNxrtGearRepaint	boolean
XmNxrtGearScrollBarHoriz	widget_ref
XmNxrtGearScrollBarVert	widget_ref
XmNxrtGearSelectionPolicy	integer
XmNxrtGearSelectionRestrictions	integer
XmNxrtGearTextEditor	widget
XmNxrtGearTextHorizClipPolicy	integer
XmNxrtGearTreeData	string
XmNxrtGearWidthPreferredCount	integer
XmNxrtGearWidthResizePolicy	integer
XmNxrtGearWidthSizingPolicy	integer
XmNxrtGearWidthVirtual[a]	integer

a. Get only.

Reasons
XmNxrtGearActivateCallback
XmNxrtGearColumnMoveCallback
XmNxrtGearFolderStateCallback
XmNxrtGearNodeCreateCallback
XmNxrtGearNodeEnterCellCallback
XmNxrtGearNodeImportCallback
XmNxrtGearNodeMoveCallback
XmNxrtGearNodeValidateCallback
XmNxrtGearPrintCallback
XmNxrtGearSelectionCallback

D.6.1 XrtNode Resources

UIL Argument Name	Argument Type
XmNeditable	boolean
XmNxrtGearNodeStyle[a]	widget_ref
XmNxrtGearSelected[a]	boolean

a. Get only.

D.6.2 XrtFolderNode Resources

UIL Argument Name	Argument Type
XmNeditable	boolean
XmNxrtGearFolderState	integer

D.6.3 XrtColumn Resources

UIL Argument Name	Argument Type
XmNbackground	pixel
XmNxrtGearFontListSelected	font_table
XmNxrtGearLabelAlignment	integer
XmNxrtGearMarginLeft	integer

UIL Argument Name	Argument Type
XmNxrtGearMarginRight	integer
XmNxrtGearNodeAlignment	integer
XmNxrtGearWidthMaximum	integer
XmNxrtGearWidthMinimum	integer

D.6.4 XrtNodeStyle Resources

UIL Argument Name	Argument Type
XmNbackground	pixel
XmNxrtGearBackgroundSelected	color
XmNxrtGearFolderShortcutShow	boolean
XmNxrtGearForegroundSelected	color
XmNxrtGearIconClosed	icon
XmNxrtGearIconClosedSelected	icon
XmNxrtGearIconItem	icon
XmNxrtGearIconItemSelected	icon
XmNxrtGearIconOpen	icon
XmNxrtGearIconOpenSelected	icon
XmNxrtGearLineColor	color
XmNxrtGearLineStyle	integer
XmNxrtGearLineWidth	integer
XmNxrtGearShortcutColor	pixel
XmNxrtGearShortcutShow	boolean
XmNxrtGearShortcutSize	integer

D.7 Progress Resources

UIL Argument Name	Argument Type
XmNxrtGearBarColor	color
XmNxrtGearIcon	icon
XmNxrtGearLabelFormat	string
XmNxrtGearLabelShow	boolean
XmNxrtGearProgressTimeOut	integer
XmNxrtGearProgressType	integer

Reasons
XmNxrtGearLabelFormatCallback
XmNxrtGearProgressCallback

D.8 Tab Button Resources

UIL Argument Name	Argument Type
XmNxrtGearAnchorSide	integer
XmNxrtGearCornerSize	integer
XmNxrtGearLabelPixmap	icon
XmNxrtGearLabelType	integer
XmNxrtGearLayoutSpacing	integer
XmNxrtGearMask	pixmap
XmNxrtGearPixmapRotation	integer
XmNxrtGearStringLayout	integer
XmNxrtGearStringRotation	integer
XmNxrtGearStyle	integer

D.9 Tab Manager Resources

UIL Argument Name	Argument Type
XmNxrtGearActivePageNumber	integer
XmNxrtGearActivePageWidget	integer
XmNxrtGearArrowDisplayPolicy	integer
XmNxrtGearArrowHeight	integer
XmNxrtGearArrowWidth	integer
XmNxrtGearChildType	integer
XmNxrtGearFirstVisibleTab	integer
XmNxrtGearLastVisibleTab	integer
XmNxrtGearPageAttachPolicy	integer
XmNxrtGearPageNumber	integer
XmNxrtGearPageWidget	widget_ref

UIL Argument Name	Argument Type
XmNxrtGearShadowThickness	integer
XmNxrtGearTabResize	boolean
XmNxrtGearTabRotation	integer
XmNxrtGearTabSide	integer
XmNxrtGearTabSpacing	integer
XmNxrtGearTabStretch	boolean
XmNxrtGearWidgetPosition	integer

Reasons
XmNxrtGearPageCallback
XmNxrtGearResizeCallback

D.10 Toolbar Resources

UIL Argument Name	Argument Type
XmNxrtGearBottomSpace	integer
XmNxrtGearLeftSpace	integer
XmNxrtGearPosition	integer
XmNxrtGearRightSpace	integer
XmNxrtGearTopSpace	integer

D.11 Truncated Constant Definitions

The following XRT/gear constant definitions have been truncated to fit into the UIL 31-character limit:

- `XRTGEAR_UNSPECIFIED_ACTIVE_PAGE_NUMBER` becomes `XRTGEAR_UNSPECIFIED_ACTIVE_PAGE`
- `XRTGEAR_UNSPECIFIED_PAGE_POSITION` becomes `XRTGEAR_UNSPECIFIED_PAGE_POS`

- XRTGEAR_GLABEL_LAYOUT_SPACING_DEFAULT becomes XRTGEAR_GLABEL_LAYOUT_SPACING

D.12 Paned Window Resources

UIL Argument Name	Argument Type
XmNxrtGearMarginHeight	Dimension
XmNxrtGearMarginWidth	Dimension
XmNxrtGearOrientation	unsigned char
XmNxrtGearRecomputeSize	Boolean
XmNxrtGearSashHeight	Dimension
XmNxrtGearSashIndent	Position
XmNxrtGearSashShadowThickness	Dimension
XmNxrtGearSashWidth	Dimension
XmNxrtGearSeparatorShow	Boolean
XmNxrtGearSpacing	Dimension
XmNxrtGearAllowResize	Boolean
XmNxrtGearMinimum	Dimension
XmNxrtGearMaximum	Dimension
XmNxrtGearPositionIndex	short
XmNxrtGearSkipAdjust	Boolean

Reasons
XmNxrtGearPaneResizeCallback

E

Sample Code

Examples
Demos

This appendix provides an overview of the sample code included with XRT/gear. XRT/gear provides two types of programming samples–examples and demos. Examples are short programs that illustrate *specific* features. Demos are more complete applications that illustrate *several features* working together. Most common XRT/gear programming tasks are covered in the examples and demos.

E.1 Examples

Example programs are located in *$XRTHOME/src/gear/examples*:

aligner.c	Displays an Aligner widget.
combo.c	Displays a Combo Box widget.
complex_cpp.cxx	Creates XRT/gear widgets and manipulates their resources; written in C++.
label.c	Displays an Enhanced Label widget.
multicol.c	Displays an Outliner widget containing multiple columns with labels.
outliner.c	Creates an Outliner widget displaying the data in a specified file.
paned.c	Shows the features of the paned window widget and color editor object.
progress.c	Shows the features of the Progress widget.
pushb.c	Displays an Enhanced Pushbutton widget.
racecar.c	Displays a Progress widget, illustrating the use of icons.
simple_cpp.cxx	Creates XRT/gear widgets; written in C++.
tabs.c	Displays Tab Manager and Tab Button widgets.
toggle.c	Displays an Enhanced Toggle widget.
toolbar.c	Displays a Toolbar widget.

E.2 Demos

Demo applications are located in *$XRTHOME/src/gear/demos*. There is a directory for each demo.

fileviewer	An application that uses the Outliner widget to display and modify your system file hierarchy.
sampler	An application that shows how many of the XRT/gear widgets work.

F

Simulating the Windows 95 Look

This appendix provides tips on how to give your XRT/gear widgets the appearance of Microsoft Windows 95 components.

General Tips

- Use the MS Sans Serif font, if this font is available on your machine. If this font is not available, use one of the alternative appearance fonts, such as 7-point Arial or 9-point Times New Roman.

Combo Box

- Set `XmNxrtGearArrowType` to its default value of `XRTGEAR_ARROW_WEDGE`.
- Set `XmNxrtGearAlignment` to its default value of `XRTGEAR_MENU_ALIGN_BOTH`.

Enhanced Toggle

- Set `XmNxrtGearIndicator` to `XRTGEAR_INDICATOR_CHECK`.

Outliner

- Set `XmNxrtGearShortcutSize` to 7.
- Set `XmNxrtGearBorderType` to `XRTGEAR_BORDER_FRAME_OUT`.
- Set `XmNxrtGearShadowThickness` to 1.
- Set `XmNxrtGearBorderLocation` to its default value of `XRTGEAR_BORDER_EACH_LABEL_SEPARATED_FROM_NODES`.
- Set `XmNbottomShadowColor` to black. This, combined with the frame out border type, will make your Outliner borders look like those used by Windows Explorer. To change your scrollbars, retrieve their widget IDs by getting the values of `XmNxrtGearScrollBarHoriz` and `XmNxrtGearScrollBarVert`.
- Use the default value of the NodeStyle object's `XmNxrtGearLineStyle` resource, which is `XRTGEAR_LINESTYLE_DOTTED`.
- Set the NodeStyle object's `XmNxrtGearIconOpen` resource to the closed folder icon (used by `XmNxrtGearIconClosed` and `XmNxrtGearIconClosedSelected`). In Windows 95, the closed folder icon is used to denote an unselected folder, regardless of whether its children are visible.

- For a more accurate simulation of the Windows 95 look, flip the folder pixmaps used by `XmNxrtGearIconClosed`, `XmNxrtGearIconClosedSelected`, `XmNxrtGearIconOpen` and `XmNxrtGearIconOpenSelected`. In Windows 95, the "tab" appears on the top left of the folder icon. (The open folder and closed folder icons are located in the *$XRTHOME/src/icons* directory, and are named *folder_open.xpm* and *folder_closed.xpm*.)

Progress

- Set `XmNxrtGearProgressType` to `XRTGEAR_PROGRESS_DISCRETE_BAR`.

Tab Manager

- Set `XmNxrtGearStyle` to its default value of `XRTGEAR_STYLE_ROUNDED`.
- Set `XmNxrtGearCornerSize` to 2.

Index

A

activate callbacks 144
Activate() action 71, 209
actual height and width 110
adding help text 270
adding Widget Tips 270
Aligner 1, 7, 81, 82
  alignment 85
  and Motif class hierarchy 83
  and XmSeparator 83
  baseline alignment 85
  column spacing 83
  constraint resource description 85, 87
  example 82
  label and control spacing 83
  label orientation 84
  margin 83
  position of child widgets 85
  resizing 85
  resource description 87
  row spacing 83
  widget anatomy 81
  widget hierarchy 81
alignment of Progress label 231
ANSI C, compatibility with XRT/gear 11
arm pixmap 20
  clip mask 21
ArmAndActivate() action 71
authorizing
  shared libraries 11
authorizing a program 10
automatic node sorting 108

B

bar display in Progress 231
baseline alignment 85
border
  type 19
  width 19
button
  button press pixmap 20
  colors 35
  indicators 33
  shape 54
  states 32
  type 17, 52

C

C++
  compatibility with XRT/gear 11
  using with XRT/gear 11
Cancel() action 255
CancelEdit() 209
ChangeState() action 209
changing current page 46
child widget list 271
clip mask 17, 21, 34, 53
clipping Outliner text 115
Color Editor 2, 8
colors
  and Outliner 113
column
  controlling display 124
  creating 120
  introduction 118
  label color 124
  label fonts 124
  label height 125
  labels 123
  margin 126
  node and label alignment 124
  ordering 121
  resizing 128
  selected 122
  selection callbacks 122
  width 125
column spacing 83
Combo Box 1, 8, 241, 254
  actions 255, 267
  arrow 245
  arrow widget 242
  creating 254
  customized arrow 246
  display 245
  example 241
  highlight font and searching pick list 244
  menu alignment 248
  menu callbacks 245
  menu processing 243
  methods reference 254, 265
  modifying the pick list 243
  pick list 243
  resource description 250, 262
  searching the pick list 249
  spacing 248

text field 244
translations 255, 267
user interaction 242, 248
comments on product 4
CommitEdit() 209
compiling and authorizing a program 10
controlling column display 124
controlling repainting 150
controlling tab display 48
controlling tree display 110
corner size 57
creating a node style 129
creating columns 120
creating subtrees 100
creating tree from string 101
current button state 35
current tab 46
cursor display 117
cursor movement 117
customizing nodes 129

D

data file format 93, 98
special characters 98
data type listing 323
default node style 130
defining button indicators 33
defining help translations 276
Delete() action 210
demo programs, discussion of 364
destroy procedure 292
disabling Widget Tips 274
Disarm() action 71
display callbacks 272
display delay 273
display side specification 48
displaying pixmaps 17, 55
distance between string and pixmap 19, 55
double buffering 149
drag and drop 141
Drag() action 210

E

Edit() 210
Editable 214
enabling help 270
Enhanced Label 1, 7, 15
border type 19
border width 19
button type 17
displaying pixmaps 17
distance between string and pixmap 19
example of 15
rotation 18
specifying a pixmap 17
string and pixmap positioning 20
using a clip mask 17
Enhanced Pushbutton 1, 7, 15
arm pixmap 20
clip mask 21
button type 17
displaying pixmaps 17
distance between string and pixmap 19
example of 16
resource description 23
rotation 18
specifying a pixmap 17
string and pixmap positioning 20
using a clip mask 17
Enhanced Toggle 1, 7
and Motif class hierarchy 32
button colors 35
button indicators 33
button states 32
clipping indicator pixmaps 34
current button state 35
defining button indicators 33
example of 31
multi-state switches 34
resource description 35
setting button state 35
state change callbacks 34
state change methods 35
value changed callbacks not used 32
environment variables, XRTHOME 8
example programs, listing of 363
executable, compiling and authorizing 10

F

filling entire side with tabs 50
folder node 95
folder states
specifying 139
fonts
and Outliner 113
and Progress 230
highlighting and Combo Box picklist 244

G

GoTo() action 210
GUI builder usage 11, 351

H

height of tree 110
help balloons, adding 269
help text, see Widget Tips
Help() 267
Help() action 71
highlight font
and pick list 244

I

Icon Collection 1
Icon Library 8
icons 131
  also see XrtGearIcon
  creating from XPM format 132
  spacing 132
indenting nodes 113
internationalization 12
item 95

L

Label 214
label
  alignment 85
  and control spacing 83
  for Outliner column 123
  for Progress widget 225
  orientation 84
label format callbacks 230
label, see Enhanced Label
lines and shortcuts 133
linking
  shared libraries 11
list of child widgets 271
list of visible nodes 114
list, see XrtList
locating events 144

M

manually displaying help text 275
manually removing help text 275
margin 112
  for Progress 232
  for toolbar 75
  of tabs 50, 56
maximum value 225
Menu() action 255
Microsoft Windows 95 365
minimum value 225
movement callbacks 107
moving nodes 107
MultiActivate() action 71
MultiArm() action 72
multiple columns, see column
multi-state switches 34

N

New Features In XRT/gear 3.0 2
NextTabGroup() 267
NoAction() action 211
node
  alignment in column 124
  automatic sorting 108
  change callbacks 104
  current 104
  customizing 129
  default style 130
  defining node style 131
  definition 95
  folder states 139
  getting children 104
  indenting 113
  label 95, 129
  label and resource files 120
  label types 118
  movement callbacks 107
  moving 107
  retrieving selected list 140
  selection behavior 136
  selection callbacks 140
  sorting 108
  specifying comparison procedure 109
  text clipping 115
  topmost visible 114
  user-defined data 135
  vertical space 113
  visibility 114
  visible list 114
node area 95
Node object 95
node style 129
  creating 129
  default 130
  defining for node 131
  icons 131
  lines and shortcuts 133
  resources 130
NodeFolder object 95

O

Online Support Resources 4
on-off switches 32
ordering columns 121
orientation
  of Aligner 84
  of Toolbar 74
Outliner 1, 7
  activate callbacks 144
  as tree root node 95
  column, see column
  concepts 95
  controlling repainting 150
  creating subtrees 100
  creating tree from data file 93
  creating tree from string 101
  cursor display 117
  data file format 93, 98
  drag and drop 141
  folder states 139
  getting started 91

icons 131
lines and shortcuts 133
locating events 144
margins 112
node change callbacks 104
node label types 118
node visibility 114
overview 91
performance issues 149
printing widget 146
resize callbacks 116
resizing 116
restructuring the tree 107
scrollbars 141
selection behavior 136
selection callbacks 140
setting current node 104
sorting nodes 108
text clipping 115
traversal convenience methods 105
traversing the tree 103
tree data file in resource file 100
tree display 110
user interaction 135
user-defined data 135

P

page changed callbacks 46
page shadow colors 57
paging arrow buttons in Tab Manager 51
PanedWindow 1, 8
PanedWindow example 260
performance issues in Outliner 149
pick list 243
  and highlight font 244
  searching 249
pixmap
  for button press 20
  positioning 20
  print attributes 284
  printing 284
  rotation 18, 57
  snapshots 283
pop-up help balloons 269
PostScript
  output for Outliner widgets 146
  specifying substitution font 147
preferred height and width 110
PrevTabGroup() 268
Print callback 148
printing
  Outliner widgets 146
  pixmap 284
  print callbacks 148
  widgets 283
  XImage snapshots 285
product feedback 4
Progress 1, 8
  bar color 233
  bar count 232
  bar display 231
  bar icons 233
  bar spacing 233
  bar type 231
  current value 225
  destroying 228
  formatted string 229
  introduction 223
  label alignment 231
  label color 231
  label display 228
  label font 230
  label formatting 229
  label range 225
  LabelFormat callbacks 230
  margin height and width 232
  maximum value 225
  minimum value 225
  number of bars 232
  percentage of task completed 226
  procedures and methods 238
  recomputing size to fit label 231
  resource description 233
  sample program 223
  suppressing label display 229
  TimeOut callbacks 227
  timeouts 226
pushbutton, see Enhanced Pushbutton

R

repainting
  Outliner 150
resize callbacks 58, 116
ResizeColumn() action 211
resizing a column 128
resizing Aligner 85
resizing Outliner 116
resource files
  syntax 351
  use of 351
resource type/class listing 339
restructuring the tree 107
retrieving child widget list 271
retrieving selected node list 140
rotation 18, 50, 56
row spacing 83

S

sample code 363
SashAction(commit) 268
SashAction(key, increment, direction) 268
SashAction(move) 268
SashAction(start) 268
scrollbars 141

scrolling in Tab Manager 51
Select() action 211
Selected 214
selected columns 122
selected node list 140
selection behavior 136
selection callbacks 140
separators and Aligner 83
setting button state 35
setting current node 104
shared libraries
  linking and authorizing 11
shortcuts 133
snapshots of widgets 283
specifying a pixmap 17, 53
specifying folder states 139
specifying label type 17
specifying node comparison procedure 109
specifying pixmap print attributes 284
state change callbacks 34
state change methods 35
string
  also see XrtString
  converting to tree 101
string and pixmap positioning 20
string rotation 18, 56
Style 215
subtrees 100
suppressing label display 229
suppressing page changes 46

T

tab
  anchoring 53
  resizing 49
  rotation 50
  spacing 49
  stretching 50
  visibility 49
Tab Button 1, 7
  and Motif class hierarchy 45
  attaching to pages automatically 47
  button shape 54
  button style 54
  button type 52
  corner rounding 57
  corner size 57
  current tab 46
  displaying pixmaps 55
  distance between string and pixmap 55
  example 44
  margin control 56
  page shadow colors 57
  resource description 59
  rotation 56
  specifying a pixmap 53
  specifying colors 57
  tab anchoring 53
  using a clip mask 53
Tab Manager 1, 7
  and Motif class hierarchy 45
  automatic Tab Button attachment 47
  changing current page 46
  constraint resource description 65
  constraint resources 52
  controlling tab display 48
  dynamic tab creation 47
  example 44
  margins 50
  page changed callbacks 46
  paging arrow buttons 51
  resize callbacks 58
  resource description 61
  setting current page 46
  setting current tab 46
  setting display side 48
  suppressing page changes 46
  tab resizing 49
  tab rotation 50
  tab spacing 49
  tab stretching 50
  tab visibility 49
technical support 3
text clipping 115
text rotation 18
timeout callbacks 227
timeouts 226
toggle button, see Enhanced Toggle
Toolbar 1, 7, 73
  and Motif class hierarchy 74
  child spacing 75
  child widget position 76
  constraint resource description 75, 77
  example 73
  margin 75
  orientation 74
  resource description 76
  spacing 75
topmost visible node 114
tracking cursor movement 117
translations
  Widget Tips 276
traversing the tree 103
  convenience methods 105
tree
  colors and fonts 113
  creating from data file 93
  display 110
  drag and drop 141
  height and width 110
  lines and shortcuts 133
  list of visible nodes 114
  margins 112
  movement callbacks 107
  moving nodes 107
  node visibility 114
  restructuring 107
  selection behavior 136

sorting 108
traversal 103
vertical space between nodes 113
tree data file format 98

U

UIL
reasons 355, 360
user interaction 248
default user interaction 248
UserData 215
user-defined data 135
using resource files 11

V

vertical space between nodes 113
vertical Toolbar 74
virtual height and width 110
visibility of nodes 114
visible tabs 49

W

widget
introduction 1
Motif class hierarchy 9
widget help, see Widget Tips
Widget Snapshot 1, 8, 283
introduction 283
pixmap snapshots 283
printing a pixmap 284
resource description 285
specifying print attributes 284
widget synopses 16, 32, 45, 74, 82, 97, 225
Widget Tips 1, 8, 269
adding help balloons to widgets 269
adding to a widget 270
border style 271
colors 271
defining help translations 276
disabling 274
display callbacks 272
display delay 273
enabling 270
fontlist 271
help balloon appearance 271
help display groups 273
introduction 269
manually displaying help text 275
manually removing help text 275
resource description 276
retrieving list of child widgets 271
widget anatomy 269
XImage snapshots 285
width of tree 110
Windows 95 365

X

XBM pixmap format 17, 21, 53
XDestroyImage() 285
XFreePixmap() 283
XImage snapshots 285
XmCreateXrtAligner() 89
XmCreateXrtColumn() 197
XmCreateXrtGearComboBox() 254
XmCreateXrtGLabel() 28
XmCreateXrtNode() 185
XmCreateXrtNodeFolder() 186
XmCreateXrtNodeStyle() 196
XmCreateXrtOutliner() 178
XmCreateXrtPanedWindow() 265
XmCreateXrtProgress() 238
XmCreateXrtProgressShellSimple() 228, 239
XmCreateXrtPushButton() 28
XmCreateXrtTabButton() 68
XmCreateXrtTabManager() 69
XmCreateXrtToggleButton() 41
XmCreateXrtToolbar() 78
XmIsXrtAligner() 90
XmIsXrtColumn() 197
XmIsXrtGearComboBox() 254
XmIsXrtGLabel() 28
XmIsXrtNode() 186
XmIsXrtNodeFolder() 186
XmIsXrtNodeStyle() 196
XmIsXrtOutliner() 178
XmIsXrtPanedWindow() 266
XmIsXrtProgress() 239
XmIsXrtPushButton() 28
XmIsXrtTabButton() 69
XmIsXrtTabManager() 69
XmIsXrtToggleButton() 41
XmIsXrtToolbar() 79
XmNactivateCallback 71
XmNarmCallback 36, 71, 72
XmNarmPixmap 21
XmNbackground 21, 24, 36, 59, 113, 115, 124, 131, 151, 158, 171, 173, 231, 233
XmNbottomShadowColor 365
XmNdestroyCallback 210
XmNdisarmCallback 36, 71
XmNeditable 155, 170, 173
XmNfontList 22, 24, 36, 59, 111, 112, 124, 125, 131, 157, 165, 169, 172, 174, 230, 234, 243
XmNforeground 22, 24, 36, 59, 124, 131, 133, 157, 172, 174, 231
XmNheight 22, 24, 37, 60, 110, 125, 157, 172, 174
XmNhelpCallback 71, 267
XmNhighlightColor 115, 158
XmNhighlightThickness 63, 111, 113, 115, 116, 158, 163
XmNlabelPixmap 17

XmNlabelString 22, 24, 37, 60
XmNmarginBottom 56
XmNmarginHeight 22, 25, 38, 56, 60, 232, 235
XmNmarginLeft 56
XmNmarginRight 56
XmNmarginTop 56
XmNmarginWidth 22, 25, 38, 56, 60, 232
XmNmaximum 225, 227, 228, 229, 234, 235, 236, 237
XmNminimum 225, 229, 234, 235, 237, 239
XmNmultiClick 71, 72
XmNrecomputeSize 231
XmNset 32
XmNshadowThickness 19, 22, 23
XmNstringDirection 51, 246
XmNuserData 23, 25, 39, 61, 135, 171
XmNValue 250
XmNvalue 224, 225, 226, 227, 228, 229, 234, 236, 237, 239
XmNvalueChangedCallback 32
XmNwidth 23, 25, 39, 61, 110, 125, 126, 128, 129, 169, 175, 176, 211
XmNx 23, 25, 39, 61, 176
XmNxrtGearActivateCallback 144, 151, 329
XmNxrtGearActive 20, 23
XmNxrtGearActivePageNumber 46, 61
XmNxrtGearActivePageWidget 46, 61
XmNxrtGearActiveTabWidget 46, 62
XmNxrtGearAlignBaseline 85, 88
XmNxrtGearAlignerAlignment 85, 87, 323
XmNxrtGearAlignment 248, 250, 276, 365
XmNxrtGearAllowResize 264
XmNxrtGearAnchorSide 53, 59, 324
XmNxrtGearArmMask 21, 24
XmNxrtGearArmPixmap 20, 24
XmNxrtGearArrow 247, 251
XmNxrtGearArrowClass 247, 251
XmNxrtGearArrowDisplayPolicy 51, 62, 326
XmNxrtGearArrowHeight 51, 62
XmNxrtGearArrowType 245, 251, 324, 365
XmNxrtGearArrowWidth 51, 62
XmNxrtGearAttachPolicy 324
XmNxrtGearAutoComplete 250
XmNxrtGearAutoSort 108, 143, 151, 159, 163, 193
XmNxrtGearBackground 271, 277
XmNxrtGearBackgroundSelected 171
XmNxrtGearBarColor 233
XmNxrtGearBarCount 232, 234
XmNxrtGearBarSpacing 233, 234
XmNxrtGearBorderLocation 126, 151, 324, 365
XmNxrtGearBorderType 19, 21, 127, 152, 183, 324, 365
XmNxrtGearBottomSpace 75, 77
XmNxrtGearCharWidthChar 169
XmNxrtGearChildType 52, 65, 325
XmNxrtGearColumnLabelsShow 152
XmNxrtGearColumnList 152
XmNxrtGearColumnMoveAllow 145, 152
XmNxrtGearColumnMoveCallback 146, 153
XmNxrtGearColumnOrderList 122, 153
XmNxrtGearColumnPreferredVisibleCount 153
XmNxrtGearColumnSelected 122, 153, 211
XmNxrtGearColumnSelectionCallback 122, 153, 325
XmNxrtGearColumnSortOrderList 159
XmNxrtGearColumnSpacing 83, 87
XmNxrtGearCornerSize 54, 59, 62, 366
XmNxrtGearCursor 117, 154
XmNxrtGearCursorResizeWidth 117, 154
XmNxrtGearCursorTracking 117, 154
XmNxrtGearCursorTrackingCallback 117, 118, 154, 326
XmNxrtGearDelay 273, 277
XmNxrtGearDisplayCallback 272, 277, 326
XmNxrtGearDoubleBuffer 150, 155, 234
XmNxrtGearEditColumn 155
XmNxrtGearEditNode 155
XmNxrtGearEditor 156
XmNxrtGearFirstVisibleTab 49, 63
XmNxrtGearFolderState 171, 327
XmNxrtGearFolderStateCallback 156, 186, 219
XmNxrtGearFolderStateMask 139, 156, 171, 186, 218
XmNxrtGearFontList 271, 277
XmNxrtGearFontListSelected 124, 131, 172, 174
XmNxrtGearForeground 271, 277
XmNxrtGearForegroundSelected 172
XmNxrtGearGroup 274, 277
XmNxrtGearHeightPreferredCount 111, 157
XmNxrtGearHeightResizePolicy 116, 157, 337
XmNxrtGearHeightSizingPolicy 111, 116, 157, 335
XmNxrtGearHeightVirtual 111, 158
XmNxrtGearHighlightThickness 50
XmNxrtGearIcon 234
XmNxrtGearIconClosed 131, 172, 365
XmNxrtGearIconClosedSelected 131, 172, 365
XmNxrtGearIconItem 131, 172
XmNxrtGearIconItemSelected 131, 172
XmNxrtGearIconOpen 131, 172, 365
XmNxrtGearIconOpenSelected 131, 173
XmNxrtGearImportCallback 144, 158
XmNxrtGearIndicator 33, 37, 328, 365
XmNxrtGearIndicatorMaskList 34, 37
XmNxrtGearIndicatorPixmapList 37
XmNxrtGearLabel 96, 102, 118, 119, 120, 129, 171, 174, 189, 225, 234
XmNxrtGearLabelAlignment 124, 125, 174, 231, 234, 323
XmNxrtGearLabelFormat 229, 234
XmNxrtGearLabelFormatCallback 230, 235
XmNxrtGearLabelPixmap 17, 21, 22, 24, 53, 60
XmNxrtGearLabelShow 229, 235
XmNxrtGearLabelType 17, 20, 22, 24, 52, 60, 337
XmNxrtGearLastVisibleTab 49, 63
XmNxrtGearLayoutSpacing 19, 22, 24, 55, 60, 132, 173
XmNxrtGearLeftSpace 75, 77
XmNxrtGearLineColor 133, 173
XmNxrtGearLineStyle 134, 173, 328, 365
XmNxrtGearLineWidth 134, 173
XmNxrtGearMarginHeight 50, 63, 75, 77, 83, 87,

112, 126, 159, 262
XmNxrtGearMarginLeft 126, 174
XmNxrtGearMarginRight 126, 174
XmNxrtGearMarginWidth 50, 63, 75, 77, 87, 112, 126, 159, 262
XmNxrtGearMask 17, 23, 25, 53, 60, 132
XmNxrtGearMaximum 264
XmNxrtGearMenuAlignment 328
XmNxrtGearMenuCallback 245, 251, 256, 329
XmNxrtGearMenuList 243, 252
XmNxrtGearMinimum 264
XmNxrtGearNodeAlignment 124, 125, 174, 323
XmNxrtGearNodeCheckForDuplicates 150, 159
XmNxrtGearNodeChildList 104, 159, 171, 191, 192
XmNxrtGearNodeCompareProc 108, 109, 159, 186
XmNxrtGearNodeCreateCallback 159, 210, 329
XmNxrtGearNodeCurrent 104, 139, 158, 160
XmNxrtGearNodeCurrentChangedCallback 104, 160, 330
XmNxrtGearNodeEnterCellCallback 129, 161
XmNxrtGearNodeFolderStateCallback 330
XmNxrtGearNodeHeight 111, 115, 135, 157, 162
XmNxrtGearNodeHeightPreferredCount 162
XmNxrtGearNodeHighlightOnDrag 162
XmNxrtGearNodeImportCallback 331
XmNxrtGearNodeImportPolicy 142, 162, 327
XmNxrtGearNodeIndent 113, 162, 173
XmNxrtGearNodeIndentPolicy 327
XmNxrtGearNodeMoveCallback 108, 109, 162, 330
XmNxrtGearNodeMovePolicy 141, 163, 329
XmNxrtGearNodeShowWhole 163
XmNxrtGearNodeSpacing 111, 113, 114, 116, 163
XmNxrtGearNodeStyle 130, 131, 163
XmNxrtGearNodeStyleDefault 130, 163
XmNxrtGearNodeStyleList 131, 163, 171, 215
XmNxrtGearNodeTop 114, 163, 164
XmNxrtGearNodeValidateCallback 129
XmNxrtGearNodeVisibleItemCount 252
XmNxrtGearNodeVisibleList 114, 163, 164
XmNxrtGearNumStates 34, 38
XmNxrtGearOrientation 74, 77, 84, 87, 262
XmNxrtGearPageAttachPolicy 47, 62
XmNxrtGearPageCallback 46, 63, 323, 331
XmNxrtGearPageNumber 52, 66
XmNxrtGearPageWidget 52, 57, 66
XmNxrtGearPaneResizeCallback 263
XmNxrtGearPickList 242, 243, 252
XmNxrtGearPickListIndex 243, 252
XmNxrtGearPixmapRotation 18, 23, 25, 50, 60, 334
XmNxrtGearPosition 76, 77, 85, 88, 264
XmNxrtGearPrintCallback 148, 164, 332
XmNxrtGearProgressCallback 226, 236, 332
XmNxrtGearProgressTimeOut 226, 236
XmNxrtGearProgressType 231, 233, 237, 332, 366
XmNxrtGearPSFontMapList 147, 165, 333
XmNxrtGearRecomputeSize 263
XmNxrtGearRepaint 150, 165
XmNxrtGearResizeCallback 58, 64, 116, 128, 166, 174, 211, 327, 334
XmNxrtGearResizeHeight 85, 88
XmNxrtGearResizePolicy 128
XmNxrtGearResizeWidth 85, 88
XmNxrtGearRightSpace 76, 77
XmNxrtGearRowSpacing 83, 87
XmNxrtGearSashHeight 263
XmNxrtGearSashIndent 263
XmNxrtGearSashShadowThickness 263
XmNxrtGearSashWidth 263
XmNxrtGearScrollBarHoriz 141, 166, 365
XmNxrtGearScrollBarHorizDisplayPolicy 141, 166, 326
XmNxrtGearScrollBarVert 141, 166, 365
XmNxrtGearScrollBarVertDisplayPolicy 141, 166, 326
XmNxrtGearSearchPolicy 244, 249, 252, 334
XmNxrtGearSelectColor 35, 38
XmNxrtGearSelected 140, 171, 175
XmNxrtGearSelectedBackground 131
XmNxrtGearSelectedForeground 131
XmNxrtGearSelectedNodeList 139, 140, 167
XmNxrtGearSelectionCallback 140, 167, 334
XmNxrtGearSelectionPolicy 136, 167, 168, 198, 211, 334
XmNxrtGearSelectionRestrictions 138, 168, 335
XmNxrtGearSeparatorShow 263
XmNxrtGearShadowThickness 64, 128, 152, 168, 211, 278, 365
XmNxrtGearShortcutShow 134, 173
XmNxrtGearShortcutSize 134, 173, 365
XmNxrtGearSkipAdjust 264
XmNxrtGearSnapNoShapeClip 286
XmNxrtGearSorted 304
XmNxrtGearSpacing 248, 252, 264
XmNxrtGearState 32, 34, 38, 336
XmNxrtGearStateChangedCallback 32, 34, 38, 336
XmNxrtGearStringCopy() 298
XmNxrtGearStringLayout 19, 23, 25, 50, 55, 61, 336
XmNxrtGearStringRotation 18, 23, 25, 50, 56, 61, 334
XmNxrtGearStyle 54, 61, 336, 366
XmNxrtGearTabResize 49, 51, 64
XmNxrtGearTabRotation 50, 56, 57, 60, 61, 65, 334
XmNxrtGearTabSide 48, 51, 65, 336
XmNxrtGearTabSpacing 49, 65
XmNxrtGearTabStretch 50, 51, 65
XmNxrtGearText 270, 275, 278
XmNxrtGearTextEditor 168
XmNxrtGearTextField 244, 252
XmNxrtGearTextHorizClipPolicy 169, 336
XmNxrtGearTipsBorderType 271, 278, 324
XmNxrtGearTopSpace 75, 77
XmNxrtGearTreeData 100, 169
XmNxrtGearUnselectColor 35, 38
XmNxrtGearValuePercent 226, 229, 234, 237
XmNxrtGearVisibleItemCount 248
XmNxrtGearWidgetPosition 66
XmNxrtGearWidthChar 112, 125, 157, 170, 176
XmNxrtGearWidthMaximum 128, 175

XmNxrtGearWidthMinimum 128, 175
XmNxrtGearWidthPreferredCount 112, 125, 129, 170, 175, 176
XmNxrtGearWidthResizePolicy 116, 129, 170, 175, 325, 337
XmNxrtGearWidthSizingPolicy 112, 116, 125, 150, 169, 170, 175, 176, 335
XmNxrtGearWidthVirtual 111, 170
XmNxrtNodeIndent 169
XmNy 23, 25, 39, 61
XmPushButtonCallbackStruct 71
XmSeparator 83
XmToggleButtonCallbackStruct 36
XPM pixmap format 17, 21, 53
XRT/gear
  Aligner 81, 82
  and C++ 11
  and UIL 353
  data types 323
  Enhanced Label 15
  Enhanced Pushbutton 15
  Enhanced Toggle 31
  icons for widgets 1, 8
  internationalization 12
  introduction 1
  Motif class hierarchy 9
  Outliner 91
  overview 7
  resource types/classes 339
  Tab Button 44
  Tab Manager 44
  Toolbar 73
  using resource files 351
  Widget Snapshot 283
  Widget Tips 269
  widgets 7
XRT/gear utilities 8
xrt_auth program
  flags 10
  use of 10
XrtArm() action 72
XrtColorEditor 300
XrtColorEditor object 300
XrtColorEditorChildType 325
XrtColumn object 120
XrtGearAction 323
XrtGearAlignerAlignment 323
XrtGearAlignment 323
XrtGearAnchorSide 324
XrtGearArrowType 324
XrtGearAttachPolicy 324
XrtGearBorderLocation 324
XrtGearBorderType 324
XrtGearChildType 325
XrtGearCmpDoubles() 295, 301
XrtGearCmpFloats() 295, 301
XrtGearCmpInts() 295, 301
XrtGearCmpStrings() 295, 302
XrtGearCmpStringsIgnoreCase() 295, 302
XrtGearColorEditorAddCancelCallback() 318
XrtGearColorEditorAddOKCallback() 318
XrtGearColorEditorAllocateColor() 318
XrtGearColorEditorCreate() 313
XrtGearColorEditorDestroy() 313
XrtGearColorEditorGetChild() 314
XrtGearColorEditorGetColorName() 316
XrtGearColorEditorGetEditTypes() 318
XrtGearColorEditorGetHSBLabels() 317
XrtGearColorEditorGetHSBValues() 317
XrtGearColorEditorGetName() 314
XrtGearColorEditorGetRGBLabels() 316
XrtGearColorEditorGetRGBValues() 317
XrtGearColorEditorHide() 314
XrtGearColorEditorIsShowing() 314
XrtGearColorEditorSetCloseColorProc() 318
XrtGearColorEditorSetColorList() 315
XrtGearColorEditorSetColorName() 315
XrtGearColorEditorSetEditTypes() 317
XrtGearColorEditorSetHSBLabels() 316
XrtGearColorEditorSetHSBValues() 316
XrtGearColorEditorSetRGBLabels() 315
XrtGearColorEditorSetRGBValues() 315
XrtGearColorEditorShow() 314
XrtGearColumnFromXEvent() 145, 151, 197
XrtGearColumnFromXY() 197
XrtGearColumnMoveCallbackStruct 153, 325
XrtGearColumnResizePolicy 325
XrtGearColumnSelectionCallbackStruct 154, 325
XrtGearCursorTrackingCallbackStruct 155, 326
XrtGearCvtStringToXmString() 178, 310
XrtGearCvtXmStringToString() 178, 310
XrtGearDirection 326
XrtGearDisplayCallbackStruct 277, 326
XrtGearDisplayPolicy 326
XrtGearEditType 326
XrtGearEnhancedSizeCallbackStruct 175, 327
XrtGearFolderState 327
XrtGearIcon 300, 327
  methods reference 319
XrtGearIconGetSize() 319
XrtGearIconLoadFromXpmData() 133, 319
XrtGearIconLoadFromXpmFile() 133, 319
XrtGearImportPolicy 327
XrtGearIndentPolicy 327
XrtGearIndicator 328
XrtGearIsColorEditor() 313
XrtGearIsNode() 215
XrtGearIsNodeFolder() 219
XrtGearLineStyle 328
XrtGearListAdd() 291, 302
XrtGearListAppend() 96, 291, 302
XrtGearListChangedCallbackStruct 328
XrtGearListCreate() 96, 290, 291, 303
XrtGearListDelete() 97, 291, 303, 308
XrtGearListDeleteAll() 303, 308
XrtGearListDestroy() 290, 292, 303
XrtGearListDestroyAll() 291, 292
XrtGearListDestroyObject() 292, 303
XrtGearListDestroyXtPointer() 292, 304
XrtGearListDuplicate() 290, 304

XrtGearListFind() 293, 295, 304
XrtGearListGetCompareProc() 293, 302, 304, 307, 308
XrtGearListGetItem() 293, 305
XrtGearListGetItemCount() 97, 294, 305
XrtGearListGetItems() 294, 305
XrtGearListGetItemSize() 294, 302, 305, 306
XrtGearListGetUserData() 297, 305
XrtGearListInsertAfter() 291, 306
XrtGearListInsertBefore(291
XrtGearListInsertBefore() 291, 306
XrtGearListIsList() 292, 298, 306
XrtGearListIsSorted() 296, 306
XrtGearListLookupItem() 293, 307
XrtGearListMoveItemsAfter() 307
XrtGearListMoveItemsBefore() 296, 307
XrtGearListRemove() 291, 303, 307
XrtGearListRemoveAll() 291, 303, 308
XrtGearListSearchBackward() 294
XrtGearListSearchForward() 294, 308
XrtGearListSearchReverse() 308
XrtGearListSetChangedProc() 328
XrtGearListSetCompareProc() 293, 295, 296, 301, 302, 309
XrtGearListSetDestroyProc() 292, 304, 309
XrtGearListSetItem() 294, 309
XrtGearListSetUserData() 297, 309, 310
XrtGearListSort() 97, 108, 296, 310
XrtGearLocation 328
XrtGearMenuAlignment 328
XrtGearMenuCallbackStruct 251, 329
XrtGearMovePolicy 329
XrtGearMrmInitialize() 28, 41, 69, 79, 90, 178
XrtGearNodeActivateCallbackStruct 151, 329
XrtGearNodeChangeFolderState() 156, 186, 219
XrtGearNodeCmpLabels() 186
XrtGearNodeCmpPositionsInTree() 187
XrtGearNodeConstructPath() 187
XrtGearNodeCreateCallbackStruct 160, 329
XrtGearNodeCreateLW() 215
XrtGearNodeCreateLWTreeFromBuffer() 219
XrtGearNodeCreateLWTreeFromStream() 220
XrtGearNodeCreateTreeFromBuffer() 101, 119, 187
XrtGearNodeCreateTreeFromStream() 99, 100, 103, 119, 142, 160, 188
XrtGearNodeCurrentChangedCallbackStruct 161, 330
XrtGearNodeCvtLabelToString() 189
XrtGearNodeCvtStringToLabel() 119, 189
XrtGearNodeDestroy() 108, 215, 221
XrtGearNodeEnterCellCallbackStruct 330
XrtGearNodeFolderStateCallbackStruct 156, 330
XrtGearNodeFromXEvent() 151, 328
XrtGearNodeFromXY() 328
XrtGearNodeGetChild() 105, 190
XrtGearNodeGetChildren() 221
XrtGearNodeGetFirstInTree() 105, 190
XrtGearNodeGetFirstNonCollapsedInTree() 105, 190
XrtGearNodeGetFolderState() 221
XrtGearNodeGetHeight() 215
XrtGearNodeGetLabel() 216
XrtGearNodeGetLastInTree() 105, 190
XrtGearNodeGetLastNonCollapsedInTree() 105, 190
XrtGearNodeGetLevel() 105, 191
XrtGearNodeGetName() 216
XrtGearNodeGetNextInTree() 105, 191
XrtGearNodeGetNextNonCollapsedInTree() 106, 191
XrtGearNodeGetNodeStyle() 216
XrtGearNodeGetNumChildren() 106, 191
XrtGearNodeGetParent() 216
XrtGearNodeGetPreviousInTree() 106, 192
XrtGearNodeGetPreviousNonCollapsedInTree() 106, 192
XrtGearNodeGetSelected() 216
XrtGearNodeGetUserData() 217
XrtGearNodeGetVisibility() 106, 114, 192
XrtGearNodeGetWidgetParent() 106, 192
XrtGearNodeGetWidth() 217
XrtGearNodeImportCallbackStruct 158
XrtGearNodeIsAncestor() 193
XrtGearNodeIsInCollapsedBranch() 106, 193
XrtGearNodeMakeVisible() 114, 193
XrtGearNodeMoveCallbackStruct 162, 330
XrtGearNodeSearchTree() 326
XrtGearNodeSetEditable() 217
XrtGearNodeSetLabel() 217
XrtGearNodeSetNodeStyle() 217
XrtGearNodeSetParent() 107, 193
XrtGearNodeSetSelected() 217
XrtGearNodeSetUserData() 218
XrtGearNodeSetValuesOnTree 194
XrtGearNodeSetValuesOnTree() 194
XrtGearNodeSortTree() 108, 194
XrtGearNodeStyleFromName() 131, 196
XrtGearNodeTraverseTo() 104, 160, 194
XrtGearNodeValidateCallbackStruct 331
XrtGearNodeWriteTreeToBuffer() 103, 194
XrtGearNodeWriteTreeToFile(103
XrtGearNodeWriteTreeToFile() 102
XrtGearNodeWriteTreeToStream 102, 103
XrtGearNodeWriteTreeToStream() 195
XrtGearOutlinerCancelEdit() 178
XrtGearOutlinerCommitEdit() 179
XrtGearOutlinerDrawPS() 13, 179
XrtGearOutlinerImportCallbackStruct 331
XrtGearOutlinerMoveColumns() 121, 122, 146, 153, 179
XrtGearOutlinerNodeFromXEvent() 145, 180
XrtGearOutlinerNodeFromXY() 180
XrtGearOutlinerTraverseToCell() 181
XrtGearOutlinerVaDrawPS() 13, 146, 181
XrtGearPageCallbackStruct 63, 331
XrtGearPaneResizeCallbackStruct 332
XrtGearPrintCallbackStruct 332
XrtGearPrintCallbackStruct structure 149
XrtGearProgressCalcTimeElapsed() 239
XrtGearProgressCalcTimeToCompletion() 239
XrtGearProgressCalcTimeTotal() 239
XrtGearProgressCallbackStruct 230, 332
XrtGearProgressType 233, 332

XrtGearPSFontMap 333
XrtGearReason 333
XrtGearResizeCallbackStruct 64, 166, 334
XrtGearRotation 334
XrtGearSearchPolicy 334
XrtGearSelectCallbackStruct 167, 334
XrtGearSelectionPolicy 334
XrtGearSelectionRestrictions 335
XrtGearSetDestroyProc() 303, 308
XrtGearSetXmStringConverter() 29, 41, 69, 79, 90
XrtGearSizingPolicy 335
XrtGearSnapArg 335
XrtGearSnapDrawArg 335
XrtGearSnapGetWidgetImage() 285, 286
XrtGearSnapGetWidgetPixmap() 283, 286
XrtGearSnapPrintPixmap() 285, 286, 335
XrtGearSnapVaPrintPixmap() 284, 287
XrtGearState 336
XrtGearStateChangedCallbackStruct 38, 336
XrtGearStringCompare() 299, 310
XrtGearStringCopy() 311
XrtGearStringCreate() 297, 311
XrtGearStringCreateCharString() 96, 298, 311
XrtGearStringCreateIconString() 96, 298, 311
XrtGearStringCreateXmString() 96, 298, 312
XrtGearStringDestroy() 298, 312
XrtGearStringGetValue() 299, 312
XrtGearStringIsCharString() 298, 312
XrtGearStringIsIconString() 298, 312
XrtGearStringIsXmString() 298, 313
XrtGearStringLayout 336
XrtGearStringSetValue() 299, 313
XrtGearStyle 336
XrtGearTabManagerMakeTabVisible() 49, 70
XrtGearTabRotation 336
XrtGearTabSide 336
XrtGearTextClipPolicy 336
XrtGearTipsActionDisplay() 276, 278
XrtGearTipsActionRemove() 276, 278
XrtGearTipsAddCallback() 273, 278
XrtGearTipsAddCallbacks() 279
XrtGearTipsDisplayHelp() 275, 279
XrtGearTipsEnable() 270, 279
XrtGearTipsGetValues() 270, 280
XrtGearTipsGetWidgetList() 271, 276, 280
XrtGearTipsRemoveAllCallbacks() 280
XrtGearTipsRemoveCallback() 273, 280
XrtGearTipsRemoveCallbacks() 281
XrtGearTipsRemoveHelp() 275, 281
XrtGearTipsSetValues() 270, 281
XrtGearTipsTrackingEvent() 281
XrtGearTipsVaGetValues() 270, 282
XrtGearTipsVaSetValues() 270, 272, 282
XrtGearToggleButtonGetState() 35, 42
XrtGearToggleButtonSetState() 35, 42
XrtGearType 337
XrtGearWidgetResizePolicy 337
XRTHOME environment variable 8
XrtList
  adding items 290
  adding sorted items 291
  checking for existence 292
  comparing items 295
  copying 290
  creating 290
  destroy procedure 292
  destroying 290
  destroying item memory 291
  introduction 290
  item count 294
  methods reference 301
  moving items 296
  removing items 291
  retrieving entire array 294
  retrieving item by index 293
  sample code 290
  searching 292
  sorting 296
XrtList object 96
XrtPanedWindow widget 259
XrtPanedWindowGetPaneList() 266
XrtString
  comparing strings 299
  copying 298
  creating 297
  destroying 298
  determining string type 298
  getting the string value 299
  methods reference 310
  setting the string value 299
XrtString object 96
XtDestroyWidget 108
XtDestroyWidget() 210
XtName() 103, 187, 195
XtVaSetValues() 181, 272