XRT/gauge™ for Motif

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 1.1



Software Development Productivity

260 King Street East
Toronto, Ontario, Canada M5A 4L5
(416) 594-1026
www.klgroup.com

December 1999

Part No: KLVGM11

Copyright © 1999-2000 by KL Group Inc. All Rights Reserved.

KL Group, the KL Group logo, XRT, XRT/3d, XRT/care, XRT/field, XRT/gauge, 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
Assumptions 2
Typographical Conventions Used in this Manual 2
Related Documents 2
Technical Support 2
Product Feedback and Announcements 3

Part I: Using XRT/gauge

1 Introduction to XRT/gauge 7
1.1 What is XRT/gauge? 7
1.2 XRT/gauge and the Motif Class Hierarchy 9
1.3 Environment Variables 9
1.4 Compiling and Authorizing a Program 11
1.5 Using Resource Files and GUI Builders 12
1.6 Using C++ with XRT/gauge 12
1.7 XRT/gauge Internationalization 13
1.8 Common Elements in XRT/gauge 14
Macros 14
Setting Fallback Resources 14
Batching Resource Updates 14
Debug Statements 15
Centering Text Objects 15

2 Circular Gauge and Circular Dial Widgets 17
2.1 Description of the Widgets 17
Elements Common to the Circular Gauge and Circular Dial 18
Circular Gauge 20
Circular Dial 20
2.2 Objects in Circular Gauges and Circular Dials 21
Center Objects (Circular Gauge) 21

Circular Scales (Gauge) or Circular Scale (Dial) 21
Frames 22
Marker (Dial) 22
Needles (Circular Gauge) 22
Ranges 23
Switch 23
State 23
Text Areas 24
Ticks (Gauge) 24
2.3 Translations and Actions 24
2.4 Sample Code 25
2.5 Widget Synopses 27
2.6 Resource Reference 28
Resources Common to Both Circular Widgets 28
Circular Gauge–Additional Resources 30
Circular Dial–Additional Resources 30
2.7 Procedures and Methods Reference 32

3 Linear Gauge and Linear Dial Widgets 35

3.1 Description of the Widgets 35
Elements Common to the Linear Gauge and Linear Dial . . . 36
Linear Gauge Objects 37
Linear Dial Objects 38
3.2 Sample Program 39
3.3 Translations and Actions 40
3.4 Widget Synopses 40
3.5 Resource Reference 41
Common Resources 41
Linear Gauge–Additional Resources 43
Linear Dial–Additional Resources 43
3.6 Procedures and Methods Reference 44

4 The LEDIcon Widget 47

4.1 Introduction 47
4.2 LEDIcon's Resources 48
4.3 Translations and Actions 50
4.4 Code Sample 50
4.5 Widget Synopsis 50
4.6 Resource Reference 51
4.7 Procedures and Methods Reference 55

5 The LEDNumerical Widget 57
5.1 Introduction 57
5.2 The Value of an LED Numerical Widget 58
Setting Decimal Places 58
5.3 Coloring the Widget and the Segments 58
5.4 Dimensioning an LED Numerical widget 58
5.5 Clock Displays 60
5.6 Slanting the Numerals 60
5.7 Callbacks 61
5.8 Code Sample 61
5.9 Widget Synopsis 61
5.10 Resource Reference 62
5.11 Procedures and Methods Reference 65

6 The Percent Widget 67
6.1 The Percent Widget 67
6.2 Choosing a Scale 69
6.3 Label Extents 70
6.4 Placing a Label on an Item 71
6.5 Notes on the Percent Widget's Resources 71
6.6 Translations and Callbacks 72
6.7 Widget Synopsis 72
6.8 Resource Reference 72
6.9 Procedures and Methods Reference 75

7 Center Object 77
7.1 XrtGaCenter 77
7.2 Notes on the Center Object's Resources 77
7.3 Sample Code 78
7.4 Resource Reference 79
7.5 Procedures and Methods Reference 79

8 Circular Scale Object 81
8.1 XrtGaCircularScale 81
8.2 Angles In a Circular Scale 81
8.3 Notes on Circular Scale Resources 82
Specifying the Direction of Travel 83
Specifying Start and Stop Angles 83
Callbacks 83
8.4 Sample Code 83
8.5 Resource Reference 84

8.6 Procedures and Methods Reference 85

9 Counter Object 87

9.1 XrtGaCounter 87
9.2 Counter Resources 87
Using the Counter Object as a Widget 87
Choosing a Border and a Background Color 87
Choosing Colors 88
Choosing a Font 88
Using Images for the Counter's Background 88
Setting the Decimal Place 88
Using Transitions 88
The Reset Button 88
Setting a value for the Counter 89
9.3 Callbacks and Translations 89
9.4 Sample Code 89
9.5 Widget Synopsis 90
9.6 Resource Reference 90
9.7 Procedures and Methods Reference 93

10 Frame Object 95

10.1 XrtGaFrame 95
10.2 Notes on Setting the Frame Object's Resources 96
10.3 Sample Code 96
10.4 Resource Reference 96
10.5 Procedures and Methods Reference 98

11 Icon Object 99

11.1 XrtGaIcon 99
11.2 Notes on the Icon's Resources 99
Coloring the Icon 99
Using Images in the Icon 100
Icon Shapes 100
11.3 Sample Code 100
11.4 Resource Reference 100
11.5 Procedures and Methods Reference 101

12 Indicator Object 103

12.1 XrtGaIndicator 103
12.2 Notes on the Object's Resources 104
General Information 104

Indicator Extents 104
Reversing the Sense of a Pointer or Triangle 104
12.3 Callbacks 104
12.4 Sample Code 104
12.5 Resource Reference 105
12.6 Procedures and Methods Reference 107

13 Item Object 109
13.1 XrtGaItem 109
13.2 Notes on the Object's Resources 110
13.3 Sample Code 111
13.4 Resource Reference 111
13.5 Procedures and Methods Reference 112

14 Linear Scale Object 115
14.1 XrtGaLinearScale 115
14.2 Creating a Linear Scale 115
The Orientation and Direction of a Linear Scale 116
14.3 User Interaction 117
14.4 Sample Code 117
14.5 Resource Reference 118
14.6 Procedures and Methods Reference 119

15 Marker Object 121
15.1 XrtGaMarker 121
15.2 Notes on the Object's Attributes 122
15.3 Sample Code 122
15.4 Resource Reference 123
15.5 Procedures and Methods Reference 124

16 Needle Object 125
16.1 XmNxrtGaNeedle 125
16.2 A Note on the Needle Object's Resources 126
16.3 Code Sample 126
16.4 Resource Reference 127
16.5 Procedures and Methods Reference 128

17 Range Object 129
17.1 XrtGaRange 129
17.2 Notes on the Range Object's Resources 130
17.3 Sample Code 132

17.4 Resource Reference 133
17.5 Procedures and Methods Reference 135

18 State Object 137
18.1 XrtGaState 137
18.2 Notes on the State Object's Attributes 137
18.3 Sample Code 139
18.4 Resource Reference 140
18.5 Procedures and Methods Reference 143

19 Switch Object 145
19.1 XrtGaSwitch 145
19.2 Notes on the Switch 146
19.3 Sample Code 147
19.4 Callbacks 148
19.5 Resource Reference 148
19.6 Procedures and Methods Reference 150

20 Text Object 153
20.1 XrtGaText 153
20.2 Notes on the Text Object's Resources 153
20.3 Sample Code 154
20.4 Resource Reference 155
20.5 Procedures and Methods Reference 157

21 Tick Object 159
21.1 XrtGaTick 159
21.2 Notes on the Tick Object's Resources 160
21.3 Sample Code 162
21.4 Resource Reference 162
21.5 Procedures and Methods Reference 165

22 Programming User Interaction 167
22.1 User Interaction in XRT/gauge 167
22.2 Overview of Translations and Actions 167
22.3 Starting Widget/Object Interaction 168
22.4 Updating Widget/Object Interaction 169
Ending Widget/Object Interaction 170
22.5 Starting Translating and Zooming 170
22.6 Updating Translating and Zooming 172
Translation and Graphics Zooming 172

Translation and Zooming . . . 173
22.7 The Counter Object's Reset Action . . . 173
Mapping . . . 174
The Value Changed Callback . . . 178
Event Handling Procedures . . . 178
22.8 Window Resizing . . . 179

23 Advanced Topics . . . 181
23.1 Providing Your Own Draw Function . . . 181
The XrtGaDrawData Structure Parameters by Object . . . 184
23.2 Setting Images with XtConvertAndStore . . . 186

24 PDS Utility Library . . . 189
24.1 XrtPDSImage . . . 189
Creating an XrtPDSImage . . . 189
24.2 Registering Image Types . . . 190
Predefined Image Support . . . 190
Custom Image Support . . . 191
Writing the Image Functions . . . 192
24.3 Using Dynamic or Shared Libraries . . . 193
24.4 XrtPDSList Object . . . 193
Sample Code . . . 194
Creating and Destroying a List . . . 194
Adding and Deleting List Items . . . 194
Searching and Retrieving List Items . . . 196
List Comparison Routines . . . 199
Sorting a List . . . 200
User-Defined Data . . . 201
24.5 Procedures and Methods Reference . . . 201
XrtPDSImage Methods . . . 201
XrtPDSList Object Methods . . . 204

Part II: Appendices

A Macros and Methods . . . 217
A.1 Macros . . . 217
A.2 Methods . . . 217

B Data Types . . . 221
B.1 Enumerations . . . 221
B.2 Structures . . . 226

C Resource Types and Classes 229
C.1 Center Object 229
C.2 Circular Dial Widget 229
C.3 Circular Gauge Widget 230
C.4 Circular Scale Object 231
C.5 Counter Object 232
C.6 Frame Object 233
C.7 Icon Object 233
C.8 Indicator Object 233
C.9 Item Object 234
C.10 LEDIcon Widget 234
C.11 LEDNumerical Widget 235
C.12 Linear Dial Widget 236
C.13 Linear Gauge Widget 237
C.14 Linear Scale Object 238
C.15 Marker Object 238
C.16 Needle Object 239
C.17 Percent Widget 239
C.18 Range Object 240
C.19 State Object 240
C.20 Switch Object 241
C.21 Text Object 242
C.22 Tick Object 243

D Inherited Resources 245

E Resource File and GUI Builder Usage 247
E.1 Syntax 247
E.2 Example Code and Resource File 249
Sample Program 249

F UIL Arguments 251
F.1 Center Object Resources 251
F.2 Circular Dial Widget Resources 252
F.3 Circular Gauge Widget Resources 252
F.4 Circular Scale Object Resources 253
F.5 Counter Object Resources 254
F.6 Frame Object Resources 255
F.7 Icon Object Resources 255
F.8 Indicator Object Resources 255
F.9 Item Object Resources 256

F.10 LEDIcon Widget Resources 256
F.11 LEDNumerical Widget Resources 258
F.12 Linear Dial Widget Resources 259
F.13 Linear Gauge Widget Resources 259
F.14 Linear Scale Object Resources 260
F.15 Marker Object Resources 260
F.16 Needle Object Resources 261
F.17 Percent Widget Resources 261
F.18 Range Object Resources 262
F.19 State Object Resources 262
F.20 Switch Object Resources 263
F.21 Text Object Resources 263
F.22 Tick Object Resources 264
F.23 Truncated Constant Definitions 265

G Sample Code 267
G.1 Examples 267
G.2 Demos 268

Index 269

Preface

Introduction ■ *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/gauge adds a new set of widgets to Motif. This set consists of the following widgets:

- *CircularGauge* and *CircularDial* provide the features of a almost any kind of circular numerical display. The circular gauge measures the chosen quantity continuously by moving a pointer, either under program control or by dragging with a mouse. The dial lets end users use a mouse to position a pointer at discrete increments around the scale.
- *Linear Gauge* and *Linear Dial* supply the same functionality as their circular counterparts, but their scales are linear.
- *LED Numerical* provides a display that resembles the easy-to-read, seven-segment numerical displays found on watches, timers, and counters.
- *LED Icon* provides the type of display most commonly seen on graphic equalizers.
- The *Percent* widget presents a visual indication of the proportional size each member of a group of quantifiable entities bears to the whole.

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

XRT/gauge Objects

The object-oriented design of XRT/gauge allows reuse of widget resources. For you the user of this product, it results in a shorter learning curve because the API is simpler and easier to use. The objects are described in the next chapter.

XRT/gauge also includes *User Interaction Callbacks*, which enable you to let your users interact with the dials and gauges using the mouse.

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/gauge applications. An understanding of basic Motif concepts such as getting and setting resource values is required before continuing with this manual. See Related Documents on this page for information on Motif documentation.

Typographical Conventions Used in this Manual

`Typewriter Font`

- C language source code and examples of file contents.
- XRT/gauge 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/gauge 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/gauge. You may purchase annual XRT/care agreements from your XRT Reseller or KL Group. XRT/care ensures you will receive all XRT/gauge 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/gauge from KL Group, contact KL Group for technical support; if you purchased XRT/gauge 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/gauge 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/gauge 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/gauge, 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 4L5 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 klg-unixnews-request@klgroup.com. Visit the KL Group web site at http://www.klgroup.com for more details.

Part I

Using XRT/gauge

1

Introduction to XRT/gauge

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

1.1 What is XRT/gauge?

XRT/gauge is a collection of widgets for Motif applications that contains a group of handy analog and digital displays. There is a corresponding settable dial for each type of gauge. Like other XRT widgets, those in XRT/gauge are an expansion of the Motif widget set.

XRT/gauge Widgets

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

- *Circular Gauge* and *Circular Dial*, which provide the features of circular measuring devices and switches or needles for setting values.
- *Linear Gauge* and *Linear Dial*, which provide the features of linear measuring devices and objects such as indicators for setting values.
- A *Numeric LED*, which enables you to store and display numbers using the software equivalent of the familiar seven-segment display.
- An *Icon LED*, which is a way of showing values by turning icon objects on and off. Familiar examples are progress meters and spectrum analyzer displays on entertainment audio systems.
- A *Percent* widget, which enables you to display data items as a fraction of the whole in the form of a pie chart or a bar chart whose segments have areas proportional to the quantity that they represent.

The following list describes the objects in XRT/gauge. Each one has a chapter of its own so that its resources are easy to find.

The Objects

XRT/gauge has an object oriented design. Each widget is composed of objects, and in many cases these objects are common to more than one widget. Moreover, many resources are shared among objects. This results in a simplified structure that is easy to master and easy to program. You can create an object and set its resources, then, associate it with other objects having the same parent widget. Any values that need to change as your application runs may be changed using a call to *XtSetValues()* or *XtVaSetValues()*.

- A *Center* object belongs to a Circular Gauge widget. It is used to provide a visual indication of the center of the scale.
- A *Circular Scale* object forms the basis for both Circular Dial and Circular Gauge widgets. It holds the start and stop values assigned to the scale, and determines the angular span of the scale. Also, it controls the position and size of other objects associated with it.
- A *Counter* object can be used in Circular Gauge and Circular Dial widgets to enable you to display a count in a way that resembles a mechanical odometer. As the count increases, you can choose to have the numbers gradually roll over rather than having one number instantly replace another. Because it is an object, it can be placed in a circular gauge or a circular dial, making it possible to display the same data in both analog and digital form simultaneously, or it can display independent information, such as in a speedometer-odometer combination. *Counter* can mimic the behavior of a mechanical counter, in which numbers "roll over" from one value to the next rather than making an abrupt transition.
- A *Frame* object can be placed in Circular Gauge, Circular Dial, Linear Gauge, Linear Dial, and Percent widgets. It is used to provide a colored panel inside the widget.
- *Icon* objects are used in the LEDIcon widget as the individual elements. By manipulating their *on/off* colors, the widget may be used to display how some activity is progressing or how some value is changing.
- *Indicator* objects may be part of a Linear Gauge or Linear Dial widget. In the case of a Linear Gauge, their values may be used to mark places on the scale. No more than one indicator may be placed in a Linear Dial, where it is used as a way of showing the dial's *on/off* location.
- *Item* objects are the elements of a Percent widget.
- A *Linear Scale* object forms the basis of a Linear Gauge or a Linear Dial widget. Multiple scales may exist in a Linear Gauge, but a Linear Dial allows only one active linear scale.
- A *Marker* object may be part of a Circular Dial widget, where it is used to provide an indication of the currently selected state.
- *Needle* objects are used in the Circular Gauge widget to point to a particular value on a scale. There may be multiple needles per scale and multiple scales in the Circular Gauge widget.
- *Range* objects may be placed in Circular Gauge, Circular Dial, and Linear Gauge widgets. They provide one way of "painting" areas of the scale, as for example, a danger zone at the top end of a thermometer or a tachometer.

- *State* objects in Circular Dial widgets are used to list the allowable discrete states for a switch.
- A *Switch* object in a Circular Dial widget functions like a needle, but it is capable of being put in a discrete mode in which only certain values are allowed.
- *Text* objects may be placed in Circular Gauge, Circular Dial, Linear Gauge, Linear Dial, and Percent widgets. They provide a general-purpose labeling function.
- *Tick* objects are used in Circular Gauge, Circular Dial, Linear Gauge, and Linear Dial widgets to provide graduated markers allowing analog measurements to be inferred.

1.2 XRT/gauge and the Motif Class Hierarchy

Conceptually, XRT/gauge 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/gauge widget allows you to use resources defined by its parent widget class. For example, all XRT/gauge widgets can use resources defined for the XmPrimitive and Core widget classes.

XRT/gauge has a uniform hierarchy structure. All components derive from Core, then XmPrimitive, then one of the seven XRT/gauge widgets.

1.3 Environment Variables

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

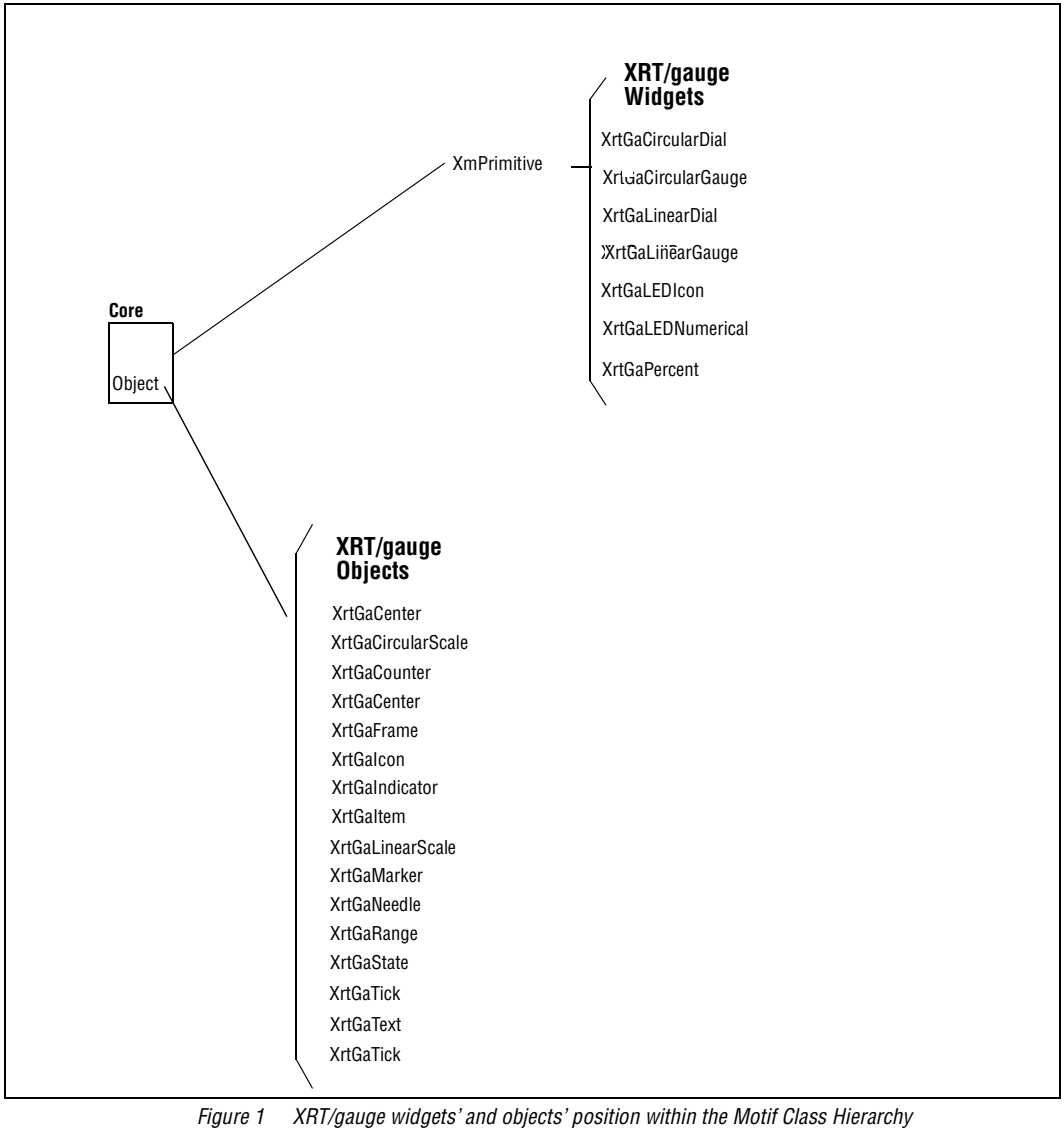


Figure 1 XRT/gauge widgets' and objects' position within the Motif Class Hierarchy

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 \
    -lxrtgauge -lpdsutil-lXm -lXpm -l Xt -X11 -lm
$(XRTHOME)/bin/xrt_auth simple
```

xrt_auth

After the XRT/gauge 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/gauge, 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 -qv executable_filename
```

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/gauge.

If your application is not successfully enabled using *xrt_auth*, XRT/gauge widgets will appear, but will display the message in Figure 2. The message will also be written to *stdout*.

```
This application is built with a disabled or an expired
time-limited version of XRT/gauge for Motif.  Please contact
KL Group Inc. or your XRT/gauge 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/gauge for Motif Programmer's Guide & Reference Manual.)
    3) That you have the XRTHOME environment variabledefined.
    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

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

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

C++ and ANSI C

The XRT/gauge 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/gauge's resources. Resource files make it easy to prototype different resource settings and quickly see the results. The section "Resource File and GUI Builder Usage" in the Appendix on page 247 provides details on the syntax of resource settings when used in resource files.

Using GUI Builders

XRT/gauge integrates with popular GUI builders. To set XRT/gauge resources from a GUI builder, use the same syntax as that used in resource files. For information on this syntax, see the section "Resource File and GUI Builder Usage" in the Appendix on page 247.

1.6 Using C++ with XRT/gauge

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

Widget	Class	Constructor Function
Circular Dial	XrtGaCircularDial	XrtGaCircularDial()
Circular Gauge	XrtGaCircularGauge	XrtGaCircularGauge()
Linear Dial	XrtGaLinearDial	XrtGaLinearDial()
Linear Gauge	XrtGaLinearGauge	XrtGaLinearGauge()
LED Icon	XrtGaLEDIcon	XrtGaLEDIcon()
LED Numerical	XrtGaLEDNumerical	XrtGaLEDNumerical()
Percent	XrtGaPercent	XrtGaPercent()

The following are the classes and constructors for the objects used by the XRT/gauge widgets:

Object	Class	Constructor Function
Center	XrtGaCenter	XrtGaCenter()
CircularScale	XrtGaCircularScale	XrtGaCircularScale()
Counter	XrtGaCounter	XrtGaCounter()
Frame	XrtGaFrame	XrtGaFrame()
Icon	XrtGaIcon	XrtGaIcon()
Indicator	XrtGaIndicator	XrtGaIndicator()
Item	XrtGaItem	XrtGaItem()
LinearScale	XrtGaLinearScale	XrtGaLinearScale()
Marker	XrtGaMarker	XrtGaMarker()
Needle	XrtGaNeedle	XrtGaNeedle()
Range	XrtGaRange	XrtGaRange()
State	XrtGaState	XrtGaState()
Switch	XrtGaSwitch	XrtGaSwitch()
Text	XrtGaText	XrtGaText()
Tick	XrtGaTick	XrtGaTick()

In addition to the above, the *XrtGauge.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 `XmNxrtGaBorderType` is *setBorderType()*.
- A method to retrieve each resource. For example, the method to get the value of `XmNxrtGaBorderType` is *getBorderType()*.
- Methods to perform common Xt functions: *addCallback()*, *destroy()*, *getDisplay()*, *getName()*, *getParent()*, *isManaged()*, *isRealized()*, *manageChild()*.

C++ examples can be found in *$XRTHOME/src/gauge/examples/simple_cpp.cxx*.

1.7 XRT/gauge Internationalization

ISO Latin1 Character Set

Any X Window font supported on your X Server may be used for the text in XRT/gauge 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).

1.8 Common Elements in XRT/gauge

1.8.1 Macros

Many XRT/gauge examples make use of two macros to reduce the amount of typing needed to define a color or to set a double precision value. Here are the definitions of these macros, located in *$XRTHOME/src/tools/mtgauge.h*:

```
#define COLOR_RES(res, color)   XtVaTypedArg, res, XmRString, \
                                        color, strlen(color) + 1

#define DVAL(value)                     XrtGaDoubleToArgVal(value)
```

1.8.2 Setting Fallback Resources

A convenient way of setting resource values is to declare them as elements of a fallback array. The following code snippet shows setting which font is to be used via an `XmFontList` resource and also setting the images that are to be used as *on* and *off* images in a linear dial.

```
static String fallback_resources[] = {
    "*.XmFontList: -*-helvetica-medium-r-normal--*-120-*-*-*-*-*-*",
    "*.XmXrtGaLinearDial.xrtGaOnImage:
                            $XRTHOME/src/icons/disk_3D_blue.xpm",
    "*.XmXrtGaLinearDial.xrtGaOffImage:
                            $XRTHOME/src/icons/disk_3D_blue2.xpm",
    NULL
};
```

Note that environment variables, such as *$XRTHOME*, are automatically assigned to their current values.

1.8.3 Batching Resource Updates

Normally resource changes take effect immediately after the *XtSetValues()* or *XtVaSetValues()* call. If you would prefer to make several changes to the widget's resources before causing a repaint, set `XmNxrtGaRepaint` to `False`. All resource changes will be batched until `XmNxrtGaRepaint` is set to `True`.

1.8.4 Debug Statements

XmNxrtGaDebug

This resource specifies whether extra warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

1.8.5 Centering Text Objects

To center Text objects horizontally within a widget, use `MtCenterText`:

```
MtCenterText(widget, textObject);
```

where `widget` is the object's parent widget and `textObject` is the Text object. It can be found in *$XRTHOME/src/tools/mtgauge.c*.

There are additional methods in *$XRTHOME/src/tools/mtgauge.c* that may be useful in application development.

2

Circular Gauge and Circular Dial Widgets

Description of the Widgets ■ *Objects in Circular Gauges and Circular Dials*
Translations and Actions ■ *Sample Code*
Widget Synopses ■ *Resource Reference*
Procedures and Methods Reference

This chapter describes the common features of the Circular Gauge and Circular Dial widgets, and provides information on their operation.

2.1 Description of the Widgets

Both Circular Gauges and Circular Dials are containers for interactive circular meters that display a current value and allow it to be measured on a scale. They are constructed using an assortment of objects that provide structured functionality to the widget as a whole. If you wish, interactions can be turned off and the component can be used simply as a display device. Either widget possesses a background color that can be thought of as a pane that fills everything within the widget's borders. You can achieve a two-color effect by providing the widget with a *frame*. In addition to coloring the widget's background, you can supply an image.

These widgets behave like analog devices. If a more exact digital readout is desired, one or more Counter objects may be placed in the widget. Figure 3 shows a circular gauge functioning as a speedometer. A counter could be added to function as an odometer or a tripometer.

You can control whether to double buffer the image and you can control the order in which objects are rendered.

You can have multiple instances of some objects a widget. In this case, the widget keeps one list for each of the object groups. The widget has a text list for keeping track of all its text objects, a range list for keeping track of the bands, called *ranges*,

that can be used to mark various regions around the scale, and a tick list to keep track of the different tick objects. Because the Circular Gauge may have more than one circular scale object, it has a list to keep track of them as well.

There is also a render list, which, if you choose to use it, lists the objects in the order in which they are to be rendered. By default, the widget has its own rendering order.

The circular scale is the reference object for a group of objects that as a whole combine to make a functioning circular gauge or circular dial. Each widget usually contains at least one circular scale. The circular scale is an object comprised of various resources that control the location of its center, its size, its minimum and maximum values, and the start and stop angles that determine the portion of a full circle filled by the scale. For instance, the needle object provides the visual indication of a particular value on the scale. The needle determines its center and its length by first referring to the origin and radius values of the circular scale. Its own length is specified as a decimal fraction of the circular scale's radius. If the circular scale changes its position, or its size changes, the needle adjusts itself accordingly, maintaining a proper center position and proportional length relative to the size of the circular scale. Note that other than being able to specify its background color, the circular scale contains no other visual information.

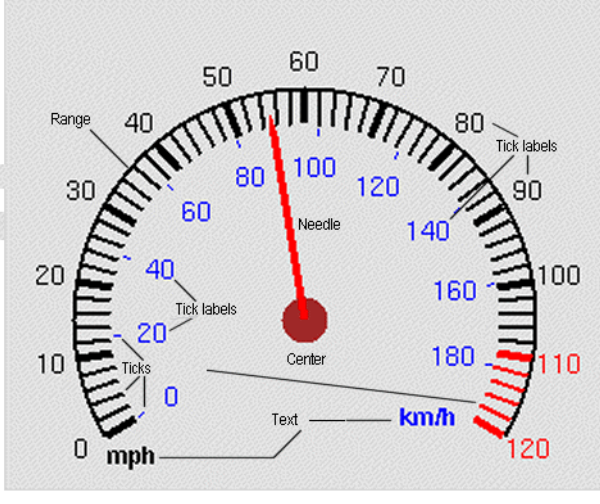


Figure 3 A Circular Gauge showing the objects used with a circular scale

A Circular Dial contains a Switch object so that it may function in a discrete mode. In this mode, a list of states defines the allowed values. There are resources for highlighting the active state, and for drawing lines from the states to their labels.

2.1.1 Elements Common to the Circular Gauge and Circular Dial

Both Circular Gauges and Circular Dials have these common elements:

Background color

This color may be applied to the widget. The background color is inherited if it is not set.

The Circular Scale

This object forms the basis for all the visible elements of the gauge or dial except the Counter object.

Range object

A Range object provides a visible indication of the scale, or provides highlighting for a portion of a circular scale. By using Range objects, an implementor can draw arcs (which may be full circles) and provide differently colored subdivisions on the circular scale.

Radius

The radius parameter sets the size of the circular scale.

Min, max

These values specify the beginning and ending values for the scale.

Direction of travel

The direction of travel is the direction in which the scale values increase, which may be clockwise or counterclockwise.

Start and stop angles

These two angles specify the compass positions marking where the *min* and *max* values are located. A circular scale positions zero degrees at due east. Angles increase in the counterclockwise direction. A start angle may be greater than a stop angle, which would be the case if a scale's *min* value begins at the twelve o'clock position and its *max* value is at the three o'clock position. In this case, the start angle is ninety degrees and the stop angle is zero degrees.

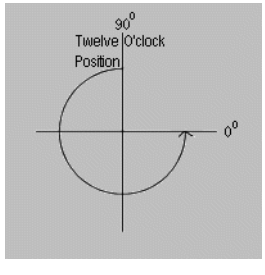


Figure 4 Angular measurement in a Circular Scale object

Ticks

Tick objects provide annotation marks on the circular scale. They provide the usual graduations that are often found on measuring devices

Frame

A frame is a rectangular portion of a circular gauge that you wish to highlight with a color. Its location is derived from its margins, which are measured from the highlight thickness inside the widget's border.

Render list

A render list is created by the widget which determines the order in which child objects are to be drawn. For example, a center object is drawn first, then a needle, so that the needle is fully visible from center to tip instead of being partially covered by the center object. There are functions for manipulating the list so that a different drawing order can be specified. By drawing a needle first, it can appear to be attached to the edge of the center object rather than beginning at the center of the circle.

2.1.2 Circular Gauge

The circular gauge widget is a container for one or more circular scales and one or more counters. In addition to the common objects just mentioned, these are also associated with a circular gauge:

- Needles – the pointers that mark the current value for the measured quantity. A circular scale can have more than one needle
- Center object – a graphical object marking the center the circular scale.

2.1.3 Circular Dial

A circular dial differs from a circular gauge in that it can switched among a collection of discrete states which form the dial's only possible values. When in this mode, the position of the dial has values that can be used in a `case` statement to decide what action to invoke. As well, a dial may be set to its continuous mode in which it takes on a continuous set of values, just like a circular gauge.

- Switch – Whereas a circular gauge has a list of needles, a dial has a *switch*. There is a maximum of one active switch per parent widget, that is, one active switch per circular dial. Creating a new switch causes it to be added to a list of switch objects. Only one of these switch objects can be active at a given time. The Circular Dial's `XmNxrtGaSwitch` resource determines which is the active switch. If there is only one active switch and if it is destroyed, the resource is set to `NULL`.
- State – The State object provides a visualization, using a line and text, of a particular position on a circular switch. In discrete mode, the list of states determines how many positions the switch can take. In continuous mode, the state simply provides annotations on the scale. Text strings label the states. The State object has positioning attributes *line inner radius*, *line outer radius*, and *offset*, which it normally gets from the switch. Each state can override these attributes if desired.
- Marker – A *marker* may be associated with a switch. A marker is used to aid in indicating the position of a switch. In a typical scenario, an end user uses a mouse click to position a switch, after which the widget updates the position of the marker.

2.2 Objects in Circular Gauges and Circular Dials

2.2.1 Center Objects (Circular Gauge)

A center object (a hub) is associated with a scale. There may be multiple centers for a scale.

You can set the center's radius and its color. Alternatively, you can specify an image for the center object.

2.2.2 Circular Scales (Gauge) or Circular Scale (Dial)

A Circular Scale has a user controllable origin position, start angle and stop angle, direction, minimum and maximum values.

Multiple scales allowed in the same circular gauge. Circular scales may be defined inside of other circular scales and share the same origin or may be completely separate from the other scales. There may be more than one circular scale per circular dial, but only one may be active at any one time. The active circular scale is the one pointed to by the Dial's `XmNxrtGaCircularScale` resource. It follows that in a circular dial only one circular scale at a time is displayed.

How Angles Are Measured in a Circular Scale

The convention for angular measurement in circular dial and gauge widgets is the same as in most texts on analytic geometry, that is, angular measurement begins with a horizontal line pointing east and increases in a counterclockwise direction. Thus, a needle pointing straight up is at 90 degrees, and straight down is at 270 degrees.

It is possible to reverse the default direction in which the *min* and *max* values begin and end on the scale. Thus, a minumum value may be attached to the scale's *start* angle. In the default situation, the scale's *min* value is attached to the scale's *start* angle.

See Chapter 8 on the Circular Scale Object for more details on the angular measurement convention in XRT/gauge.

Changing the Origin for a Circular Scale in a Circular Gauge

The following function sets a common origin for up to five circular scales in a circular gauge widget. Note the use of `XmNxrtGaXOrigin` and `XmNxrtGaYOrigin` to effect the change in origin.

```
SetCircularGaugeOrigin(cgauge, xposition, yposition)
   Widget cgauge; /* cgauge must be a Circular Gauge Widget */
    Position xposition;
    Position yposition;
   {
       XrtGaList       scale_list;
       Widget          scale_item[5];
       int             num_item;
```

```
        int             i;

        XtVaGetValues(cgauge,
                    XmNxrtGaCircularScaleList,            &scale_list,
                    NULL);

        num_item = XrtPDSListGetItemCount(scale_list);

        for (i=0; i<num_item; i++) {
            scale_item[i] = *(Widget *) XrtPDSListGetItem(scale_list, i);
        }
        scale_item[i] = NULL;

        for (i=0; i<num_item; i++) {
            if (scale_item[i] != NULL) {
                XtVaSetValues(scale_item[i],
                            XmNxrtGaXOrigin,                xposition,
                            XmNxrtGaYOrigin,                yposition,
                            NULL);
            }

        }

    }
```

2.2.3 Frames

A frame can be specified for each Circular Gauge or Circular Dial widget. It allows the user to specify a region of a different color within the widget.

The frame's shape can be rectangular or circular and its margins can be controlled. Note that depending on the value of the margins a circular frame could be elliptical in shape. Multiple frames may be created, but only one frame at a time can be active. The widget maintains a list of these frames.

2.2.4 Marker (Dial)

A marker are used to show the current position of a switch on a circular dial. It uses the circular scale's value to determine where to place itself, thus aiding the switch in providing a visual indication of the set position. If the switch itself is symmetrical, the marker may be the only way of showing the selected value.

2.2.5 Needles (Circular Gauge)

A Needle is a graphical object that lies along a radius of a Circular Scale showing a particular value on the scale. There may be multiple needles for one scale.

You can control the needle's color, length, width, and shape. The shape scales itself by using its *length* and *width* parameters.

The needle is drawn using its *shape* attribute. Basic functions exist for drawing triangles, pointers, and arrows, or you may provide your own shape.

The position of a needle, and therefore its associated value on a scale may be controlled programmatically, or via an end user action.

2.2.6 Ranges

On circular scales, ranges are rings or partial rings used to emphasize certain sections of the scale by coloring them.

Ranges are associated with a scale. There may be multiple ranges for one scale.

You can control the inner and outer radii (and thus the thickness) of the range. As well you set start-stop values, an offset to where the range starts (so ranges can exceed the start-stop values), and the color.

2.2.7 Switch

A Switch is the software equivalent to its electrical counterpart. A simple switch may have only two states, on and off, or it may posess a set of discrete states. A switch that has been put in continuous mode may be *multi-turn-enabled*, which means that it can be rotated through more than one full turn as it goes from its minimum value to its maximum value.

There is a maximum of one switch per parent widget, that is, one switch per circular dial. The active switch is pointed to by the `XmNxrtGaSwitch` resource. A Circular Dial may have more than one switch created for it, but only one of these can be active at at time. The widget maintains an internal list of the switches that have been created. Destroying the switch pointed to by the dial's `XmNxrtGaSwitch` resource causes it to be set to `NULL`.

Figure 5 shows a Switch shaped like a pointer that has four discrete states.

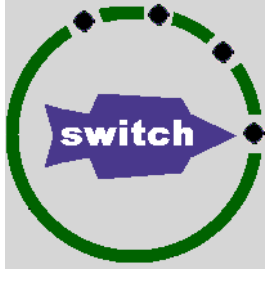


Figure 5 A Switch object with four discrete states

2.2.8 State

The *state* object is part of a circular dial. In discrete mode, the list of states determines how many positions the switch can take. In continuous mode, the state provides an indication of where the switch is pointing, and in discrete mode it indicates one of the possible positions. In discrete mode, each state may be labeled with a text string The State has attributes *line inner radius*, *line outer radius*, and *offset* to aid in the placement of the labels. Each state can override these attributes' default values if desired. See Figure 36 in Chapter 18, State Object, for more details on positioning text on states.

Users can control the state object's text string, color, angle, alignment, font, radius, and offset from the radius.

Users can also specify that states be auto-aligned, that their angles automatically be controlled (equally spaced between a start and stop angle), and that their offsets be same. Offset lines can be arrayed in a circular pattern, or always be horizontal or vertical. Their inner radius, thickness, and whether they are displayed can be controlled by the user.

2.2.9 Text Areas

A list of text areas can be provide for each circular gauge widget.

A text area has user controllable text, position, background & foreground color, images, and borders

A text area can also be bound to a switch or needle so that it can display the current value of the switch or needle in user-specified formats within the text string using the same mechanism as *printf*.

2.2.10 Ticks (Gauge)

Tick objects may be used to decorate a scale with a set of equally spaced marks, making it easier to make measurements on the scale. A scale requiring major and minor divisions would use two Tick objects. You can set the properties of one of these tick objects independently from the other, perhaps providing labels for the ticks marking the major divisions but not for the ticks marking the minor divisions.

You can set the start value, stop value, the tick increment value, and the inner and outer extents for the tick marks. If you wish, you may specify a color and a width for the tick object. If you do not set some property, the object chooses a default value.

Labels can be turned on and off for each tick object, and their font and extents can be specified.

You can set the precision of the tick marks, but be aware that reducing the precision may introduce rounding errors that may make the scale hard to read. See Chapter 2, "Tick Object" for details.

2.3 Translations and Actions

Translations for the Circular Gauge widget are:

```
<Btn1Down>:         ResetCounter(Begin) ActionStart(Needle)
<Btn1Motion>:       ActionMove(Needle)
<Btn1Up>:           ResetCounter(End) ActionEnd(Needle)
CTRL+<Btn1Down>:    ZoomStart(CircularGauge)
CTRL+<Btn1Motion>:  ZoomUpdate(CircularGauge)
CTRL+<Btn1Up>:      ZoomEnd(CircularGauge)
SHIFT+<Btn1Down>:   TranslateStart(CircularGauge)
SHIFT+<Btn1Motion>:TranslateUpdate(CircularGauge)
SHIFT+<Btn1Up>:     TranslateEnd(CircularGauge)
```

Translations for the Circular Dial are:

```
<Btn1Down>:          ResetCounter(Begin) ActionStart(Switch)
<Btn1Motion>:        ActionMove(Switch)
<Btn1Up>:            ResetCounter(End) ActionEnd(Switch)
CTRL+<Btn1Down>:     ZoomStart(CircularDial)
CTRL+<Btn1Motion>:   ZoomUpdate(CircularDial)
CTRL+<Btn1Up>:       ZoomEnd(CircularDial)
SHIFT+<Btn1Down>:    TranslateStart(CircularDial)
SHIFT+<Btn1Motion>:  TranslateUpdate(CircularDial)
SHIFT+<Btn1Up>:      TranslateEnd(CircularDial)
```

2.4 Sample Code

The following code creates a Circular Dial widget and sets some of its properties:

```
Widget cdial; /* The circular dial widget */
cdial =
       XtVaCreateManagedWidget("circular_dial",
                               xmXrtGaCircularDialWidgetClass,    form,
                               XmNleftAttachment,        XmATTACH_FORM,
                               XmNrightAttachment,       XmATTACH_FORM,
                               XmNtopAttachment,         XmATTACH_FORM,
                               XmNxrtGaHighlightSelectedState,    True,
                               XmNheight,                          350,
                               XmNwidth,                           350,
                               COLOR_RES(XmNxrtGaHighlightColor,"red"),
                               COLOR_RES(XmNbackground,       "white"),
                               NULL);
```

This creates the a widget to which a scale, a range, text and state objects may be added, resulting in the dial shown in the top part of Figure 6. The figure shows that three more circular dial widgets have been added to its lower portion to provide ways to control the upper dial's resource settings

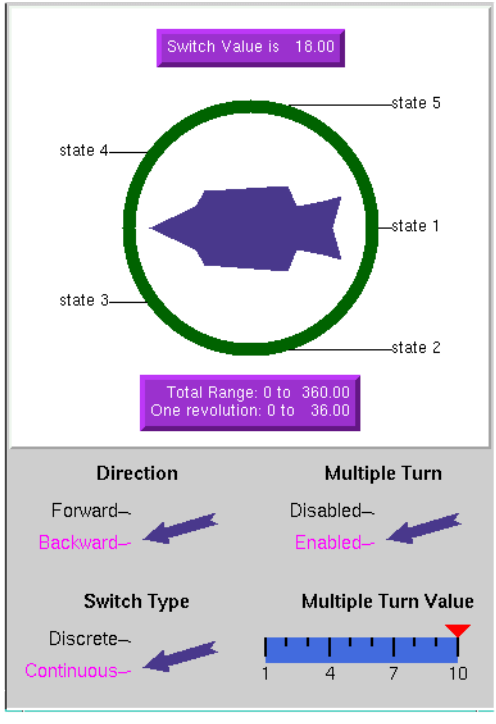


Figure 6 A sample Circular Dial

Many of the resources defined for Circular Dial are also defined for Circular Gauge. The following code creates an Circular Gauge widget:

```
Widget cgauge; /* The Circular Gauge widget */
cgauge = XtVaCreateManagedWidget("cgauge",
                                 xmXrtGaCircularGaugeWidgetClass,form,
                                     XmNheight,                    500,
                                     XmNwidth,                     300,
                                     XmNleftAttachment,    XmATTACH_FORM,
                                     XmNtopAttachment,     XmATTACH_FORM,
                                     XmNbottomAttachment,  XmATTACH_FORM,
                                     COLOR_RES(XmNbackground,   "wheat"),
                                     NULL);
```

Suitably embellished with a frame object, a text object, a center object, two range objects, a tick object, and a needle object, the code produces the following widget:

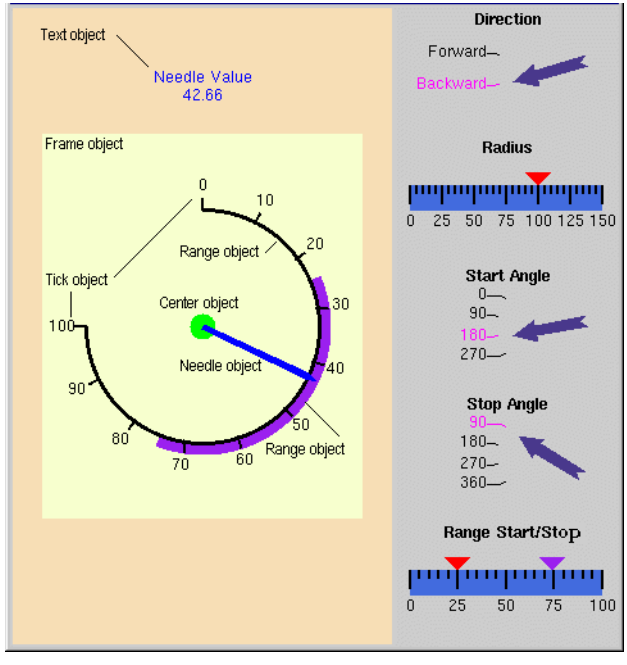


Figure 7 A sample Circular Gauge

Note that there are some circular dials and linear gauges on the right hand side of the diagram.

2.5 Widget Synopses

Circular Gauge

Include File:	*<Xm/XrtGaCircularGauge.h>*
Class Name:	XrtGaCircularGauge
Class Hierarchy:	Core → XmPrimitive → XrtGaCircularGauge
Class Pointer:	xmXrtGaCircularGaugeWidgetClass

Circular Dial

Include File:	*<Xm/XrtGaCircularDial.h>*
Class Name:	XrtGaCircularDial
Class Hierarchy:	Core → XmPrimitive → XrtGaCircularDial
Class Pointer:	xmXrtGaCircularDialWidgetClass

Since Circular Gauge and Circular Dial are widgets subclassed from XmPrimitive, a Circular Gauge can be constructed by `XtVaCreateManagedWidget()`.

For more information on the Motif widget hierarchy and where Circular Gauge and Circular Dial fit into this hierarchy, see Figure 1, XRT/gauge widgets' and objects' position within the Motif Class Hierarchy.

2.6 Resource Reference

This section lists all of the resources for the Circular Gauge and Circular Dial 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

2.6.1 Resources Common to Both Circular Widgets

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the widget. If this resource is not set then the background color of the parent widget is used

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_3D_OUT** **CSG**

Border type of the widget. `XmNshadowThickness` is used for the border width of widgets.

XmNxrtGaCounterList **XrtGaList** ***dynamic*** **G**

A list of `XrtGaCounter` objects. Each new Counter object is automatically added to its parent's counter list upon creation. This list is readonly and should not be manipulated.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDoubleBuffer **Boolean** **True** **CSG**

Decides whether or not to double buffer the drawing of the widget. Double buffering is a graphics technique that can reduce the "flickering" of an object when it is drawn or redrawn.

XmNxrtGaExposeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has repainted some portion of itself. A pointer to an `XrtGaCallbackStruct` is passed to each.

XmNxrtGaFrame **XrtGaObject** **NULL** **G**

An `XrtGaFrame` object for the widget. The type of frame is controlled by its `XmNxrtGaShape` resource. By default no frame object is created for the parent widget. If a new frame is created for the widget it will replace the old one (if it exists) and the old one is kept in a list of frames

maintained by the widget. If the frame to which this resource points is destroyed, the resource is set to NULL.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The widget's background image (if NULL, the background color is used).

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are XRTGA_IMAGE_LAYOUT_CENTERED and XRTGA_IMAGE_LAYOUT_TILED.

XmNxrtGaInteractionCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has noticed an interaction. Called during *ActionStart()*, *ActionMove()*, *ActionEnd()* and *Reset-Counter()*. A pointer to an XrtGaInteractionCallbackStruct is passed to each. See Chapter 22 for a discussion of this callback.

XmNxrtGaMapCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget *Map()* action is called. A pointer to an XrtGaMapCallbackStruct is passed to each map callback. There is no default translation that calls the *Map()* action, but one can be registered.

XmNxrtGaRangeList **XrtGaList** **NULL** **G**

A list of XrtGaRange objects (each of whose associated scale needs to be one of the circular scales on the XmNxrtGaCircularScaleList). Each new range object created is automatically added to its parents range list. This list is readonly and its contents should not be manipulated.

XmNxrtGaRenderList **XrtGaList** **dynamic** **G**

If the companion resource, XmNxrtGaUseRenderList is True, the widget will draw its object children in the order specified by the list stored in XmNxrtGaRenderList. When an object is created, it is automatically appended to the render list. Thus by default, the render list will have a list of object children in the order they were created. While this resource is read only, the user can get the pointer to the list and manipulate the list directly.

XmNxrtGaRepaint **Boolean** **True** **CSG**

This resource determines whether or not the widget is to be redrawn when any of its resources are changed. It may be worthwhile to set XmNxrtGaRepaint to False while performing numerous updates on the widget. Repainting may be turned on once all the updates are completed.

XmNxrtGaResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has resized. A pointer to an XrtGaCallbackStruct is passed to each.

XmNxrtGaTextList **XrtGaList** ***dynamic*** **G**

A list of XrtGaText objects. Each new text object is automatically added to its parent's text list when it is created. This list is readonly and its contents should not be manipulated.

XmNxrtGaTickList **XrtGaList** ***dynamic*** **G**

A list of `XrtGaTick` objects. Each new tick object created is automatically added to its parents tick list. This list is readonly and its contents should not be manipulated.

XmNxrtGaUseRenderList **Boolean** **False** **G**

If the value of this resource is `True`, the widget will draw its object children in the order specified by the list stored in `XmNxrtGaRenderList`. If this resource is `False`, the widget will draw its object children in its default order, which should be sufficient for most situations. For example, the default order for the circular gauge widget is frames, centers, ranges, ticks, needles, and text areas.

2.6.2 Circular Gauge—Additional Resources

XmNxrtGaCenterList **XrtGaList** ***dynamic*** **G**

A list of `XrtGaCenter` objects. Each new center object created is automatically added to its parents center list. This list is read only and its contents should not be manipulated.

XmNxrtGaCircularScaleList **XrtGaList** ***dynamic*** **G**

A list of `XrtGaCircularScale` objects. Each new circular scale object created is automatically added to its parents circular scale list. This list is read only and its contents should not be manipulated.

XmNxrtGaNeedleList **XrtGaList** ***dynamic*** **G**

A list of `XrtGaNeedle` objects. Each new needle object created is automatically added to its parent needle list. This list is readonly and its contents should not be manipulated.

2.6.3 Circular Dial—Additional Resources

XmNxrtGaCircularScale **XrtGaObject** **NULL** **G**

The circular scale for the widget. An internal list of circular scales is maintained for the widget. If a new circular scale is created for the widget it will replace the old one. If the object being pointed to is destroyed, this resource is set to `NULL` and nothing will be drawn in the widget (unless a counter has been created).

XmNxrtGaHighlightColor **Pixel** ***inherited*** **CSG**

Color used to highlight the selected state for a discrete switch. This defaults to the foreground color of the parent widget.

XmNxrtGaHighlightSelectedState **Boolean** **False** **CSG**

If true, the highlight color is used to highlight the line and text of the selected state for a discrete switch and for a continuous switch when the switch is no further than 2° from the state.

XmNxrtGaLineInnerRadius **double**[1] **1.0** **CSG**

Determines the inner radius of the line of each state of a switch (if the state's XmNxrtGaLineInnerRadiusUseDefault is True). This value is related to the XmNxrtGaRadiusSize value of the switch's associated circular scale. For example, a value of 0.5 means the line inner radius of the switch object is half the size of the scale's radius.

XmNxrtGaLineOuterRadius **double**[1] **1.0** **CSG**

Determines the outer radius of the line of each state of a switch (if the state's XmNxrtGaLineOuterRadiusUseDefault is True). This value is related to the XmNxrtGaRadiusSize value of the switch's associated circular scale. For example, a value of 0.5 means the line outer radius of the switch object is half the size of the scale's radius.

XmNxrtGaMarker **XrtGaObject** **NULL** **G**

Specifies a marker used for the widget that is drawn at the switch's current value. If this resource is NULL, no marker is drawn. By default, no marker object is created for the parent widget. If a new marker is created for the widget it will replace the old one (if it exists). An internal list of values is maintained by the widget. If the current marker is destroyed, this resource is set to NULL

XmNxrtGaOffsetDistance **double**[1] **1.0** **CSG**

The offset distance is used for a state if the XmNxrtGaOffsetType is XRTGA_OFFSET_TYPE_HORIZONTAL or XRTGA_OFFSET_TYPE_VERTICAL and the offset is controlled by the widget (rather than by the state[2]). This resource determines the horizontal or vertical distance from a horizontal or vertical line through the center of the widget that each state would be positioned. For each state, first a line is drawn from the inner radius to the outer radius and then to the position determined by this distance. This resource's value is based on the XmNxrtGaRadiusSize resource of the associated circular scale (a value of 1.5 means that this distance is one and a half times the radius). Note that this resource has no effect if the XmNxrtGaOffsetType is XRTGA_OFFSET_TYPE_CIRCULAR or XRTGA_OFFSET_AUTO.

XmNxrtGaOffsetType **XrtGaOffsetType** **XRTGA_OFFSET_TYPE_AUTOMATIC** **CSG**

For each state where string offsets are automatically controlled by the widget (that is, the state's XmNxrtGaOffsetUseDefault is True), this resource determines the type of offset. Offset lines can be set automatically, can radiate in a circular pattern, or can be set to be always horizontal or vertical. Possible values are

XRTGA_OFFSET_TYPE_AUTOMATIC	Default pattern.
XRTGA_OFFSET_TYPE_CIRCULAR	Radiate in a circular pattern.
XRTGA_OFFSET_TYPE_HORIZONTAL	Horizontal offsets.
XRTGA_OFFSET_TYPE_VERTICAL	Vertical offsets.

1. Reminder: When setting double values for resources, always use the function XrtGaDoubleToArgValue() or its equivalent, DVAL(). See Chapter 1, "Macros" for details.

2. Note that states are drawn the same for both discrete and continuous switches. What differs is the behavior of the switch.

XmNxrtGaStateList **XrtGaList** ***dynamic*** **G**

This list defines the allowable states. Each new state object is automatically added to its parent's state list as it is created. This list is readonly and its contents should not be manipulated.

XmNxrtGaSwitch **XrtGaObject** **NULL** **G**

The switch used by the widget. If this resource is `NULL`, only the background is drawn (unless the widget contains a counter). If a new switch is created for the widget it will replace the old one (if it exists). If the current switch is destroyed, this resource is set to `NULL`.

2.7 Procedures and Methods Reference

This section lists the Circular Gauge and Circular Dial procedures.

XmCreateXrtGaCircularDial()

Standard Motif-style function to create an `XrtGaCircularDial` widget. Returns the newly created `XrtGaCircularDial` widget or `NULL` if the widget could not be created.

```
Widget
XmCreateXrtGaCircularDial (
                    Widget      parent,
                    char        * name,
                    Arg         * arglist,
                    Cardinal    argcount)
```

Standard Motif style function to create an `XrtGaCircularDial` widget; *parent* is the parent of the `XrtGaCircularDial` widget to be created; *name* is the name to give the newly created `XrtGaCircularDial` widget; *arglist* is the list of `XrtArgVal` resource/value pairs to provide to `XrtGaCircularDial` on creation which can be `NULL`; `argcount` is the number of elements in arglist.

XmIsXrtGaCircularDial()

```
XmIsXrtGaCircularDial(w)
```

A macro that returns `True` if `w` is an `XrtGaCircularDial` widget or a subclass of an `XrtGaCircularDial` widget.

XmCreateXrtGaCircularGauge()

Standard Motif style function to create an XrtGaCircularGauge widget. Returns the newly created `XrtGaCircularGauge` widget or `NULL` if the widget could not be created.

```
Widget
XmCreateXrtGaCircularGauge (
                    Widget      parent,
                    char        * name,
                    Arg         * arglist,
                    Cardinal    argcount)
```

parent is the parent of the `XrtGaCircularGauge` widget to be created; *name* is the name to give the newly created `XrtGaCircularGauge` widget; *arglist* is the list of

`XrtArgVal` resource/value pairs to provide to `XrtGaCircularGauge` on creation which can be `NULL`; *argcount* is the number of elements in arglist.

XmIsXrtGaCircularGauge()

```
XmIsXrtGaCircularGauge(w)
```

A macro that returns True if `w` is an `XrtGaCircularGauge` widget or a subclass of an `XrtGaCircularGauge` widget.

3

Linear Gauge and Linear Dial Widgets

Description of the Widgets ■ *Sample Program*
Translations and Actions ■ *Widget Synopses*
Resource Reference ■ *Procedures and Methods Reference*

This chapter describes the common features of the Linear Gauge and Linear Dial widgets, and provides information on their operation.

3.1 Description of the Widgets

Like their circular counterparts, Linear Dials and Linear Gauges are containers for measuring or pointing devices that are arranged linearly. Each contains at least one linear scale. Use these widgets to measure and optionally control quantities whose representation naturally fits on a straight line scale. Linear scales are frequently used to measure temperature, length, area, volume, and more.

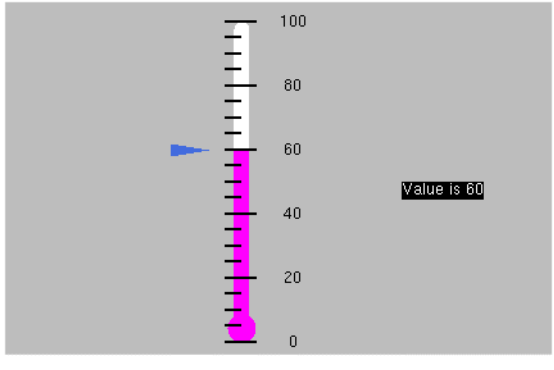


Figure 8 A basic linear dial

3.1.1 Elements Common to the Linear Gauge and Linear Dial

Background and Foreground Colors

Both widgets have settable background and foreground colors, controlled by `XmNbackground` and `XmNforeground` respectively.

The foreground color of the widget automatically becomes the foreground color of objects that are its children. For example, if you add a text object to the widget but you do not explicitly set its foreground color, the color for the text is inherited from the parent widget. The foreground color also becomes the inherited background color for some objects, such as the Switch.

Borders

The linear gauge and dial widgets have a border resource called `XmNGaBorder`. This resource controls the type of border that may be drawn around the widget. Use `XmNshadowThickness` to adjust the border's thickness. See page 41. Sample code:

```
lgauge =
    XtVaCreateManagedWidget("lgauge",
        XmNshadowThickness,     25,
        XmNxrtGaBorder,         XRTGA_BORDER_ETCHED_OUT,
        NULL);
```

XmNxrtGaDebug

Use `XmNxrtGaDebug` as a convenient way of getting the widget to print more diagnostic messages.

Rendering the Widget

The linear widgets have their preferred order of rendering, so unless `XmNxrtGaUseRenderList` is `True` this default order of rendering is used. As objects are added to the widget, their names are added to the render list in the same order as they are created. This allows you to control the order in which objects appear in the list and hence the order in which they are drawn when the widget is instructed to use its render list using `XmNxrtGaUseRenderList`. It is possible to alter the default list by changing the order in which objects are rendered to achieve some special effect. To notify the widget to use your modified render list, set the `XmNxrtGaUseRenderList` to `True`.

You can modify an existing render list:

```
Boolean     okay;
XrtGaList   renderlist;
XtVaGetValues(lgauge,
              XmNxrtGaRenderList,   &renderlist,
              NULL);
okay = XrtPDSListMoveItemsAfter(renderlist, 6, 3, 1);
```

The function *XrtPDSListMoveItemsAfter()* reorders the existing render list, moving item 6 after item 3. The last parameter indicates that just one item is being moved.

The Expose Callback

Use `XmNxrtGaExposeCallback` callback to perform extra tasks when the widget needs to be repainted. If you are using a linear gauge or dial widget in conjunction with another widget, perhaps of your own design, that must be placed on top of the linear gauge or dial, you want to ensure that it is repainted so that it isn't obscured by the linear widget.

Linear Scales

Linear Dials Linear Scale resource. Linear gauges may use one or more linear scale. In either case, the widget maintains a list of the linear scales that have been created.

These linear scales provide the measurement capabilities of a linear gauge.

Using an Image in a Linear Widget

It's possible to use an image as the background for the widget. The image can be centered in the widget or it can be used to tile the widget.

The Resize callback

Use `XmNxrtGaResizeCallback` if you need to do some repositioning of the widget after a window containing a linear gauge or dial has been resized.

Placing Text objects

If you wish to annotate various items in a widget, you can use Text objects as labels. It's possible to place as many text objects as you wish in a linear gauge or dial widget. `XmNxrtGaTextList` maintains the list of Text objects that have been created.

Tick lists

Tick objects provide a visible set of markings to facilitate reading scale values. Like the circular counterparts, linear gauges and dials can have more than one set of tick marks, usually to provide for major and minor scale markings. The objects are kept in an `XmNxrtGaTickList` where they may be referenced and manipulated.

3.1.2 Linear Gauge Objects

Adding Indicators to a Linear Gauge

An indicator is a pointer whose value provides a visible point of reference on a scale. It may be rectangular or circular, or shaped like a tank or a thermometer, or have a tiled or centered image.

A linear gauge supports multiple indicators. Thus, it is possible to mark hi-lo and current positions on a scale, and can be used in conjunction with range objects to increase a scale's readability. The linear gauge widget adds each indicator object as it is created to a read-only list. Indicators that are stored in the list may be accessed through the `XmNxrtGaIndicatorList` resource.

The Indicator's position depends on its value and on how that value relates to its associated linear scale.

Adding Ranges to a Linear Gauge

A range is a visible object that may be placed on a linear scale. The list may be accessed through `XmNxrtGaRangeList`.

There is no range list associated with a linear dial. Instead, a linear dial has its own shape, an *on* color, and an *off* color. See the section on *Indicators, On and Off Colors and Images* for details.

Batching Repaint Operations

If you are updating the widget frequently you may want to control when it is repainted. You can set `XmNxrtGaRepaint` to `False` while doing multiple updates, then turn repainting on when a period of stability is reached. This may avoid annoying flicker as a widget and its objects attempt to perform numerous updates that don't need to be seen by the end user.

3.1.3 Linear Dial Objects

A linear dial has one linear scale object and one indicator. It is easier to program than a linear gauge because it can automatically display the *on* and *off* regions of a linear scale without programming. Simply supply the scale with a value and its *on* and *off* colors, or its *on* and *off* images, and the scale automatically updates itself. To accomplish the corresponding effect with a linear gauge, a callback function has to be written to adjust two ranges, one for the *on* region and one for the *off* region.

Indicators, On and Off Colors and Images

An indicator is like the needle in a circular dial. It points to the linear dial's current value. It can be imagined as dividing the scale into two regions. The "on" region consists of all values less than or equal to the indicated value, and the "off" region consists of all values greater than the indicated value. Each region can be assigned a color or image to add another distinctive visual clue to the position of the indicator.

The `XmNxrtGaIndicator` points to the currently active indicator. The `XmNxrtGaOffColor` and `XmNxrtGaOnColor` resources control the off and on colors, while `XmNxrtGaOffImage` and `XmNxrtGaOffImage` control the off and on images.

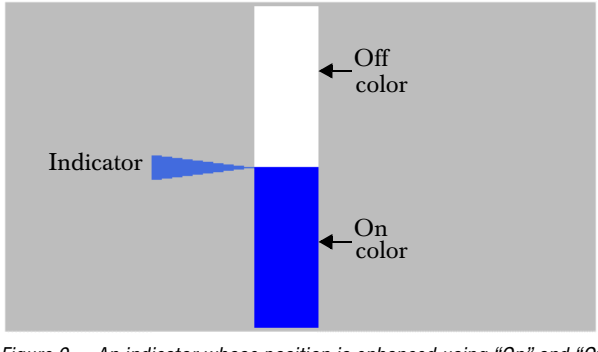


Figure 9 An indicator whose position is enhanced using "On" and "Off" colors

A linear dial has a default indicator that is created along with the widget. If you create a new indicator, the default indicator is destroyed. The XmNxrtGaIndicator resource points to this default indicator. By default its `XmNxrtGaIsShowing` resource is set to `False` so that the *off* and *on* regions can be changed without having a visible indicator.

Images can be used instead of colors to mark the on and off regions on either side of the indicator. Use `XmNxrtGaOnImage`, `XmNxrtGaOffImage` to specify the images. Here's an example of setting the appropriate images in a fallback resource file:

```
static String fallback_resources[] = {
        "*.XmXrtGaLinearGauge.xrtGaOnImage:
        $XRTHOME/src/icons/onImage.xpm",
        "*.XmXrtGaLinearGauge.xrtGaOffImage:
        $XRTHOME/src/icons/offImage.xpm",
        NULL
};
```

Choosing the Shape of the On/off Range

Use `XmNxrtGaShape` to specify the shape of the *on/ off* range. You can choose among `XRTGA_SHAPE_RECTANGLE`, `XRTGA_SHAPE_CIRCLE`, `XRTGA_SHAPE_THERMOMETER`, and `XRTGA_SHAPE_TANK`.

3.2 Sample Program

The partial program given below illustrates the construction of a Linear Gauge. First, a form is created to hold the widget, then the widget is added to it. Linear Scale, Range, and Indicator objects are added to the widget and some of their resources are set. A callback is added to the Indicator's list of functions to be called when a value changed event occurs. The function to be called, is not listed here, but it is given in *$XRTHOME/ src/ gauge/ examples/ lgauge.c.*

```
Widget              form; /* The form that holds the widget */
Widget              lgauge, lg_scale, range, indicator;
/* Create Linear Gauge */
lgauge =
    XtVaCreateManagedWidget("lgauge",
                            xmXrtGaLinearGaugeWidgetClass, form,
                            NULL);

/*Create Basic Scale*/
lg_scale = XtVaCreateWidget("lg_scale",
                            xmXrtGaLinearScaleObjectClass,   lgauge,
                            XmNxrtGaMax,                DVAL(200.0),
                            XmNwidth,                        250,
                            XmNx,                             25,
                            XmNheight,                        25,
                            XmNy,                            250,
                            NULL);

/*Create Range*/
range = XtVaCreateWidget("range",
                         xmXrtGaRangeObjectClass,lgauge,
```

```
                        XmNxrtGaScale,                  lg_scale,
                        XmNxrtGaInnerExtent,           DVAL(0.0),
                        XmNxrtGaOuterExtent,           DVAL(1.0),
                        COLOR_RES(XmNbackground,  "paleturquoise"),
                        NULL);

    /*Create Indicator*/
    indicator_type = XRTGA_INDICATOR_TYPE_TRIANGLE;
    interaction_type = XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE;
    indicator =
        XtVaCreateWidget("indicator",
                         xmXrtGaIndicatorObjectClass,          lgauge,
                         XmNxrtGaLinearScale,                lg_scale,
                         XmNwidth,                                20,
                         XmNxrtGaValue,                      DVAL(0.0),
                         XmNxrtGaIndicatorType,        indicator_type,
                         XmNxrtGaInteractionType,    interaction_type,
                         XmNxrtGaInnerExtent,             DVAL(-0.5),
                         XmNxrtGaOuterExtent,              DVAL(0.0),
                         COLOR_RES(XmNbackground,            "plum"),
                         NULL);
    XtAddCallback(indicator, XmNxrtGaValueChangedCallback, lg_ValueCB,
                  NULL);
```

3.3 Translations and Actions

Translations for the Linear Gauge widget are:

```
<Btn1Down>:         ActionStart(Needle)
<Btn1Motion>:       ActionMove(Needle)
<Btn1Up>:           ActionEnd(Needle)
CTRL+<Btn1Down>:    ZoomStart(LinearGauge)
CTRL+<Btn1Motion>:  ZoomUpdate(LinearGauge)
CTRL+<Btn1Up>:      ZoomEnd(LinearGauge)
SHIFT+<Btn1Down>:   TranslateStart(LinearGauge)
SHIFT+<Btn1Motion>:TranslateUpdate(LinearGauge)
SHIFT+<Btn1Up>:     TranslateEnd(LinearGauge)
```

Translations for the Linear Dial widget are:

```
<Btn1Down>:          ActionStart(Switch)
<Btn1Motion>:        ActionMove(Switch)
<Btn1Up>:            ActionEnd(Switch)
CTRL+<Btn1Down>:     ZoomStart(LinearDial)
CTRL+<Btn1Motion>:   ZoomUpdate(LinearDial)
CTRL+<Btn1Up>:       ZoomEnd(LinearDial)
SHIFT+<Btn1Down>:    TranslateStart(LinearDial)
SHIFT+<Btn1Motion>: TranslateUpdate(LinearDial)
SHIFT+<Btn1Up>:      TranslateEnd(LinearDial)
```

3.4 Widget Synopses

Linear Gauge

Include File: *<Xm/XrtGaLinearGauge.h>*

Class Name: XmXrtGaLinearGauge

	Class Hierarchy:	Core → XmPrimitive → XmXrtGaLinearGauge
	Class Pointer:	XmXrtGaLinearGaugeWidgetClass
Linear Dial	Include File:	*<Xm/XrtGaLinearDial.h>*
	Class Name:	XmXrtGaLinearDial
	Class Hierarchy:	Core → XmPrimitive → XmXrtGaLinearDial
	Class Pointer:	xmXrtGaLinearDialWidgetClass

For more information on the Motif widget hierarchy and where Linear Gauge and Linear Dial fit into this hierarchy, see Figure 1, "XRT/gauge widgets' and objects' position within the Motif Class Hierarchy".

3.5 Resource Reference

This section lists all of the resources for the Linear Gauge and Linear Dial 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

3.5.1 Common Resources

XmNbackground **Pixel** **inherited** **CSG**

Background color of the widget

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_3D_OUT** **CSG**

The border type of the widget. `XmNshadowThickness` is used for the border width of widgets.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDoubleBuffer **Boolean** **True** **CSG**

Whether or not to double buffer the drawing of the widget. Double buffering is a graphics technique that can reduce the "flickering" of an object when it is drawn or redrawn.

XmNxrtGaExposeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has repainted some portion of itself. A pointer to an `XrtGaCallbackStruct` is passed to each. Application programmers can use this callback to obtain notification that the widget has re-drawn itself in case they are drawing on top of the widget and need to refresh the drawing.

XmNxrtGaFrame **XrtGaObject** **NULL** **CG**

An `XrtGaFrame` object for the widget. The type of frame is controlled by the `XmNxrtGaShape` resource. See chapter 10.4 on page 96 for details. By default no frame object is created for the parent widget. If a new frame is created for the widget it will replace the old one (if it exists) and the old one is kept in a list of frames maintained by the widget. If the active frame is destroyed, this resource is set to `NULL`.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The widget's background image. If this resource is `NULL`, the background color is used.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the widget's background image. Valid values are:

`XRTGA_IMAGE_LAYOUT_CENTERED`	The image is centered on the widget. The image is clipped if it is larger than the widget.
`XRTGA_IMAGE_LAYOUT_TILED`	Multiple copies of the image are used to tile the entire area occupied by the widget.

XmNxrtGaInteractionCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has noticed an interaction. Called during *ActionStart()*, *ActionMove()*, *ActionEnd()* and *ResetCounter()*. A pointer to an `XrtGaInteractionCallbackStruct` is passed to each.

XmNxrtGaMapCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has had a `Map()` action registered. A pointer to an `XrtGaMapCallbackStruct` is passed to each map callback.

XmNxrtGaRenderList **XrtGaList** ***dynamic*** **G**

If the companion resource, `XmNxrtGaUseRenderList` is `True`, the widget draws its objects' children in the order specified by the list stored in `XmNxrtGaRenderList`. When an object is created, it is automatically appended to the render list. Thus by default, the render list has a list of object children in creation order. While this resource is read only, the user can obtain a pointer to the list and manipulate the list directly.

XmNxrtGaRepaint **Boolean** **True** **CSG**

Whether or not this widget is to be redrawn when any of its resources are changed.

XmNxrtGaResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has resized. A pointer to an `XrtGaCallbackStruct` is passed to each.

XmNxrtGaTextList **XrtGaList** **NULL** **G**

A list of `XrtGaText` objects. Each new text object created is automatically added to its parent's text list. This list is read only and its contents should not be manipulated.

XmNxrtGaTickList **XrtGaList** **NULL** **G**

A list of `XrtGaTick` objects. Each new tick object created is automatically added to its parent's tick list. This list is read only and its contents should not be manipulated.

XmNxrtGaUseRenderList **Boolean** **False** **G**

If the value of this resource is `True`, the widget draws its object children in the order specified by the list stored in `XmNxrtGaRenderList`. If this resource is `False`, the widget draws its object children in its default order, which should be sufficient for most situations.

3.5.2 Linear Gauge—Additional Resources

XmNxrtGaIndicatorList **XrtGaList** ***dynamic*** **G**

A list of the widget's `XmNxrtGaIndicator` objects. Each new indicator object created is automatically added to its parent's indicator list. This list is read only and its contents should not be manipulated.

XmNxrtGaLinearScaleList **XrtGaList** ***dynamic*** **G**

A list of `XrtGaLinearScale` objects. Each new linear scale object created is automatically added to its parent's linear scale list. This list is read-only and its contents should not be manipulated.

XmNxrtGaRangeList **XrtGaList** **NULL** **G**

A list of the widget's `XrtGaRange` objects (each of whose associated scale needs to be one of the linear scales on the `XmNxrtGaLinearScaleList`). Each new range object created is automatically added to its parents range list. This list is read only and its contents should not be manipulated.

3.5.3 Linear Dial—Additional Resources

XmNxrtGaIndicator **XrtGaObject** ***dynamic*** **CG**

An indicator whose value determines the boundary between the *on* color or image and *off* color or image. An indicator is created for the parent widget by default. If a new indicator is created for the widget it will replace the old one, and the old one is maintained in a list of indicator objects unless it is the default indicator, in which case it is destroyed. If the current active indicator is destroyed, this resource is set to `NULL`.

XmNxrtGaLinearScale **XrtGaObject** **NULL** **CG**

The widget's linear scale. If a new linear scale is created for the widget it will replace the old one and the old one is maintained in a list of linear scale objects. If the current active linear scale object is destroyed, this resource is set to `NULL` and nothing will be drawn in the widget.

XmNxrtGaOffColor **Pixel** ***inherited**** **CSG**

The off color for the linear dial.

* *The default value of* `XmNxrtGaOffColor` *is inherited from the widget's background color.*

XmNxrtGaOffImage **XrtGaImage** **NULL** **CSG**

The background image for the off section of the dial. If the value is `NULL` the off color is used.

XmNxrtGaOnColor **Pixel** ***inherited*** **CSG**

The on color for the linear dial.

* *The default value of* `XmNxrtGaOnColor` *is inherited from the widget's foreground color.*

XmNxrtGaOnImage **XrtGaImage** **NULL** **CSG**

The background image for the on section of the dial. If the value is `NULL` the on color is used.

XmNxrtGaShape **XrtGaShape** **XRTGA_SHAPE_RECTANGLE** **CSG**

The shape of the on/off range. Valid values are `XRTGA_SHAPE_RECTANGLE`, `XRTGA_SHAPE_CIRCLE`, `XRTGA_SHAPE_THERMOMETER`, and `XRTGA_SHAPE_TANK`. **Note:** the value `XRTGA_SHAPE_IMAGE` is not valid for this resource.

3.6 Procedures and Methods Reference

This section lists the Linear Dial and Linear Gauge procedures and methods.

XmCreateXrtGaLinearDial()

Creates a Linear Dial widget.

```
Widget
XmCreateXrtGaLinearDial (
    Widget parent,
    char * name,
    Arg * arglist,
    Cardinal argcount
    )
```

parent is the parent of the `XrtGaLinearDial` widget to be created; *name* is the created widget's name; *arglist* is a list of `XrtArgVal` resource/value pairs to be passed to `XrtGaLinearDial` on creation. The list can be `NULL`; *argcount* is the number of elements in *arglist.*

XmIsXrtGaLinearDial()

```
XmIsXrtGaLinearDial(w)
```

Macro that returns `True` if `w` is an `XrtGaLinearDial` widget or a subclass of an `XrtGaLinearDial` widget.

XmCreateXrtGaLinearGauge()

Creates a Linear Gauge widget.

```
Widget
XmCreateXrtGaLinearGauge (
    Widget parent,
    char * name,
    Arg * arglist,
    Cardinal argcount
    )
```

parent is the parent; *name* is the created widget's name; *arglist* is a list of `XrtArgVal` resource/value pairs to be passed to `XrtGaLinearGauge` on creation. The list can be `NULL`; *argcount* is the number of elements in *arglist*.

XmIsXrtGaLinearGauge()

```
XmIsXrtGaLinearGauge(w)
```

Macro that returns `True` if `w` is an `XrtGaLinearGauge` widget or a subclass of an `XrtGaLinearGauge` widget.

4

The LEDIcon Widget

Introduction ■ *LEDIcon's Resources*
Translations and Actions ■ *Code Sample*
Widget Synopsis ■ *Resource Reference*
Procedures and Methods Reference

4.1 Introduction

The LEDIcon widget's graphic representation resembles a software application's progress meter, or one column of an audio spectrum analyzer. The widget possesses a scale whose values are made visible through the use of a selectable number of circular or rectangular markers called *icons*. Their *on* and *off* states are distinguished by setting different foreground and background colors, or different foreground and background images. As the widget's value parameter increases more and more of the icons assume their *on* color. In a situation where the displayed quantity fluctuates rapidly and it is useful to show the maximum value attained, the highest-valued *on* icon can be instructed to remain on for a time, after which it decays through intermediated values until it reaches its now-current value.

An LEDIcon widget can assume a horizontal or vertical orientation and its scale values can be set to increase in either direction.

In the widget's numeric mode, the value at which the icons turn on, called its *LEDStartValue*, does not have to be zero but instead may be any integer. Once the widget's value parameter exceeds the LEDStartValue another icon turns on each time the value increases by one. In binary mode, the icons represent increasing powers of two. For positive numbers, the direction set to *forward*, and the orientation set to *horizontal*, the pattern begins with the units column on the far left with successively higher powers of two occupying positions in sequence to the right. Negative numbers are represented in twos complement form. For example, in binary mode -1 is represented by turning on every icon.

4.2 LEDIcon's Resources

Inherited Resources

The resources `XmNbackground`, `XmNforeground`, and `XmNxrtGaBorder` are described in the introduction. They are also listed in the resources section of this chapter.

Common Resources

The resources `XmNxrtGaDebug`, `XmNxrtGaDoubleBuffer`, and `XmNxrtGaRepaint` are also described in the introduction and are listed in the resources section of this chapter.

Setting Values

The current value for the LEDIcon widget is set using the `XmNxrtGaLEDValue` resource. In order for the widget to display the value properly, it must be an integer between the value in `XmNxrtGaLEDStartValue` and the number given by the formula

$$NumberOfIcons - LedStartValue$$

where *NumberOfIcons* is the number of icons and *LedStartValue* is the value currently set for the `XmNxrtGaLEDStartValue` resource.

There is no need to set a maximum limit for an LEDIcon widget's scale, but you must ensure that there are enough icons to display all the possible values that might be given to `XmNxrtGaLEDValue`. For example, if you need to display values that start at -5 and end at 20, you need 26 icons. If your application's GUI limits you to 10 icons and you need to display values from zero to 100, you will have to map your actual values by dividing them by 10.

Indicating the Maximum Value Attained

There are times when it is useful to be able to show, at least temporarily, the maximum value reached by a fluctuating quantity. By setting the Boolean resource `XmNxrtGaMaxIcon` to `True`, you are turning on this feature. When, in a dynamic situation, the value displayed by the widget decreases, the icon corresponding to the most recently obtained maximum will remain on for a time in milliseconds determined by `XmNxrtGaDecayRate`, after which time it will turn on the next lower icon, and so on, until the now-current value is reached. If a maximum value is reached and then quickly returns to a value a few notches down, the effect will be a visual indication of the maximum value returning to the current value by dropping through all the intermediate icons, spending a time set by `XmNxrtGaDecayRate` in each state.

Displaying Values as Binary Numbers

Besides displaying values as magnitude, with each icon corresponding to a counting number, it is possible through the use of `XmNxrtGaDisplayMode` to implement a binary-type display. When the mode of this resource is `XRTGA_DISPLAY_MODE_BINARY`, numbers are displayed in their binary equivalents. For example, if the value given by `XmNxrtGaLEDValue` is 5, the values of the first three icons on the left, assuming a horizontal orientation and a *forward* direction, would be *on*, *off*, *on*. If there were any

more icons, they would all be turned off. Changing the direction to *backward* changes the order in which the digits are displayed.

Negative numbers are displayed in twos-complement notation. That is, first form the binary equivalent of the corresponding positive number, then reverse the state of each icon, then perform the binary addition of the ensuing number and one.

`XmNxrtGaLEDStartValue` has no effect on the way binary numbers are displayed.

Setting LEDIcon's Dimensions and Layout

The widget may be oriented so that the icons form either a single row or a single column. The `XmNxrtGaOrientation` resource has two possible values, `XmHORIZONTAL` or `XmVERTICAL`. Besides choosing an orientation, use `XmNxrtGaDirection` to set the direction in which values increase. A widget in the horizontal orientation shows values increasing to the right if the `XmNxrtGaDirection` resource is set to `XRTGA_DIRECTION_FORWARD`. A widget in the vertical orientation shows values increasing in the downward direction if the `XmNxrtGaDirection` resource is set to `XRTGA_DIRECTION_BACKWARD`.

You can set the space between icons using `XmNxrtGaIconSpacing`. The setting works identically for both horizontal and vertical orientations. It simply sets the separation in pixels between one icon and the next.

The way that icons adjust themselves to the size of the widget is controlled by `XmNxrtGaSizingPolicy`. The default is `XRTGA_SIZING_POLICY_WIDGET` in which the icons occupy as much of the widget's height as possible and they adjust themselves so that they are of equal width, filling as much of the widget's width as possible.

If images are used for on and off states and the policy is `XRTGA_SIZING_POLICY_ICON`, the icons calculate a preferred size based on the largest image being used, and they pass this preferred size on to the widget. If the widget's size is too small, It makes a request to its parent to increase its size. If the parent allows the change, a larger size based on the icon's preferred size is used instead. If the widget is large enough to hold the icons in their preferred size, no size adjustment will be necessary.

If the policy is `XRTGA_SIZING_POLICY_BY_POSITION`, the position of the icons within the widget is determined by these resources:

`XmNxrtGaXOrigin`	The icon group is indented horizontally by the number of pixels given by this resource.
`XmNxrtGaYOrigin`	The icon group is indented vertically by the number of pixels given by this resource.
`XmNxrtGaIconWidth`	The common width, in pixels, for all icons. If the combined width is greater than the width of the widget, the icons will be clipped from right to left.
`XmNxrtGaIconHeight`	The common height, in pixels, for all icons. If the height is greater than the height of the widget, the icons will be clipped from bottom to top.

If the policy is XRTGA_SIZING_POLICY_CENTER, the position of the icons within the widget is determined by the XmNxrtGaIconHeight and XmNxrtGaIconWidth resources. The icons are centered within the widget.

4.3 Translations and Actions

Translations for the LEDIcon widget are:

```
<Btn1Down>:         ActionStart(Icon)
<Btn1Motion>:       ActionMove(Icon)
<Btn1Up>:           ActionEnd(Icon)
```

4.4 Code Sample

The widget maintains a list of the icons in XmNxrtGaIconList. The following code snippet shows how to get this list to find the number of icons currently created, then remove the last item.

```
Widget      led;            /* The LEDIcon widget */
Widget      icon;           /* Icon object variable
int count;                  /* Variable that holds
                               the number of items in the list */
XrtPDSList  list;

XtVaGetValues(led,
              XmNxrtGaIconList, &list,
              NULL);
count = XrtPDSListGetItemCount(list);
icon = *(XrtGaObject *)XrtPDSListGetItem(list, count - 1);
XtDestroyWidget((Widget) icon);
```

Note that destroying the icon automatically removes it from the list. Besides being able to specify images for individual icons, it is possible to provide a background image for the widget itself. Use XmNxrtGaImage for this purpose, and set the layout policy with XmNxrtGaImageLayout. Its default value is XRTGA_IMAGE_LAYOUT_CENTERED, but you can use XRTGA_IMAGE_LAYOUT_TILED instead. If you choose the tiled option, repeated instances of a small image will be used to tile the entire widget. If you choose the centered option and the image is larger than the widget, it will be clipped, not scaled to fit the available size.

Callbacks

The callbacks defined for this widget are XmNxrtGaExposeCallback, XmNxrtGaInteractionCallback, XmNxrtGaResizeCallback, and XmNxrtGaValueChangedCallback. See Chapter 22 for details.

4.5 Widget Synopsis

LEDIcon widget Include File: *<Xm/XrtGaLEDIcon.h>*

Class Name:	XmXrtGaLEDIcon
Class Hierarchy:	Core → XmPrimitive → XmXrtGaLEDIcon
Class Pointer:	xmXrtGaLEDIconWidgetClass

4.6 Resource Reference

This section lists all of the resources for the LEDIcon 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the widget. If this resource is not set then the background color of the parent widget is used

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_3D_OUT** **CSG**

The border type of the widget.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDecayRate **int** **500** **CSG**

When `XmNxrtGaMaxIcon` is `True` and the mode is magnitude, this determines the decay rate of the max icon in milliseconds.

XmNxrtGaDisplayMode **XrtGaDisplayMode** **XRTGA_DISPLAY_MODE_MAGNITUDE** **CSG**

Two types of display mode are possible: magnitude or binary. If the mode is `XRTGA_DISPLAY_MODE_MAGNITUDE`, the magnitude of the `XmNxrtGaLEDValue` resource determines how many icons are turned on. In binary mode (`XRTGA_DISPLAY_MODE_BINARY`), the bitwise representation of the value determines which icons are turned on. For example, a value of 9 (binary 1001) means that icons 0 and 3 are turned on.

XmNxrtGaDirection **XrtGaDirection** **XRTGA_DIRECTION_FORWARD** **CSG**

The direction from which *min* goes to *max*. Possible values are `XRTGA_DIRECTION_FORWARD` and `XRTGA_DIRECTION_BACKWARD`. For the LEDIcon, forward means from left to right or from bottom to top depending on the orientation, which is controlled by the `XmNxrtGaOrientation` resource.

XmNxrtGaDoubleBuffer **Boolean** **True** **CSG**

Whether or not to double-buffer the drawing of the widget. Double buffering is a graphics technique that can reduce the “flickering” of an object when it is drawn or redrawn.

XmNxrtGaExposeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has repainted some portion of itself. A pointer to an `XrtGaCallbackStruct` is passed to each

XmNxrtGaIconHeight **Dimension** **10** **CSG**

The height of each icon in the icon list.

XmNxrtGaIconList **XrtGaList** **NULL** **G**

The icon list for the widget specified in an `XrtGaList` of `XrtGaIcon` objects. If this resource is `NULL` nothing is drawn. As each new icon object is created it is automatically added to its parent's icon list. The list is read-only and its contents should not manipulated.

XmNxrtGaIconSpacing **Dimension** **2** **CSG**

The spacing in pixels between icons.

XmNxrtGaIconWidth **Dimension** **10** **CSG**

The width of each icon in the icon list. The default value depends on the sizing policy. If the sizing policy is `XRTGA_SIZING_POLICY_WIDGET`, these values are based on the widget size, the number of icons, and the icon spacing.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The widget's background image. If this value is `NULL`, the background color is used.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are `XRTGA_IMAGE_LAYOUT_CENTERED` and `XRTGA_IMAGE_LAYOUT_TILED`.

XmNxrtGaInteractionCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has noticed an interaction. Called during *ActionStart()*, *ActionMove()*, *ActionEnd()* and *ResetCounter()*. A pointer to an `XrtGaInteractionCallbackStruct` is passed to each.

XmNxrtGaInteractionType **XrtGaInteractionType** **XRTGA_INTERACTION_TYPE_NONE** **CSG**

In this widget, `XmNxrtGaInteractionType` functions like a Boolean value. The interaction mode is activated by setting either `XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE` or `XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE`. Once on, the behavior depends on `XmNxrtGaDisplayMode`.

Display Mode Setting	XmNxrtGaInteractionType Behavior
`XRTGA_DISPLAY_MODE_MAGNITUDE`	Clicking on an icon turns on this icon and all the lesser-valued icons. Dragging turns on the icon the mouse is over and all lesser-valued icons.

Display Mode Setting	XmNxrtGaInteractionType Behavior
XRTGA_DISPLAY_MODE_BINARY	Each mouse click on an icon toggles it on or off. The end user can click on the icons to form a binary number, turning icons on and off as needed. Dragging toggles on all icons between where the drag action begins and where it ends.

XmNxrtGaLEDStartValue **int** **0** **CSG**

Used to specify minimum value of LED icons. If current XmNxrtGaLEDValue is less than or equal to XmNxrtGaLEDStartValue, then all icon objects are shown using XmNxrtGaOffColor or XmNxrtGaOffImage values.

XmNxrtGaLEDValue **int** **0** **CSG**

The current value of the widget, used to turn on or off the LED icons. The number of icons in the *on* state is the value in this resource file less the *start* value. Thus if the current value of the widget is 5 and the value of the XmNxrtGaLEDStartValue resource is 3, two icons are shown in *on* state and the rest are shown in the *off* state.

XmNxrtGaMaxIcon **Boolean** **False** **CSG**

If true and the mode is magnitude, the maximum icon turn on will remain on with a decay rate of XmNxrtGaDecayRate milliseconds.

XmNxrtGaOrientation **unsigned char** **XmHORIZONTAL** **CSG**

The widget's orientation. Possible orientations are XmHORIZONTAL and XmVERTICAL. A typical example of a horizontal configuration is a progress bar. A typical example of a vertical orientation is a spectrum analyzer display.

XmNxrtGaRepaint **Boolean** **True** **CSG**

Whether or not this widget is to be redrawn when any of its resources are changed.

XmNxrtGaResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has resized. A pointer to an XrtGaCallbackStruct is passed to each.

XmNxrtGaSizingPolicy **XrtGaSizingPolicy** **XRTGA_SIZING_POLICY_WIDGET** **CSG**

This resource controls the widget size compared to the icon size. Values are

`XRTGA_SIZING_POLICY_WIDGET`	The widget's width and height determine the size of the icons and any attempt by the user to set the icon width and height is ignored.
`XRTGA_SIZING_POLICY_ICON`	The icon's height and width determine the size of the widget and any attempt by the user to set the icon width and height is ignored.
`XRTGA_SIZING_POLICY_BY_POSITION`	The `XmNxrtGaIconWidth` and `XmNxrtGaIconHeight` resources determine the size of the icons. In this case icons are clipped if there is not enough space in the widget to display them all
`XRTGA_SIZING_POLICY_CENTER`	Resources `XmNxrtGaIconWidth` and `XmNxrtGaIconHeight` determine the size of icons. Icon group is centered within the widget. Icons are clipped if there is not enough space in the widget to display them all.

XmNxrtGaSpacingHorizontal **int** **10** **CSG**

The spacing in pixels between the group of icons and the widget's right and left borders.

XmNxrtGaSpacingVertical **int** **5** **CSG**

The spacing in pixels above and below the group of icons, separating them from the parent widget by adding the specified amount of top and bottom vertical space. This resource reserves vertical space and is independent of the orientation.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

The list of callbacks called when the value changes. A pointer to an `XmNxrtGaValueChanged-CallbackStruct` is passed to each.

XmNxrtGaXOrigin **Position** **0** **CSG**

When the sizing policy is `XRTGA_SIZING_POLICY_BY_POSITION`, the *x*-coordinate of the first icon.

XmNxrtGaYOrigin **Position** **0** **CSG**

When the sizing policy is `XRTGA_SIZING_POLICY_BY_POSITION`, the *y*-coordinate of the first icon.

4.7 Procedures and Methods Reference

XmCreateXrtGaLEDIcon()

Creates an LEDIcon widget.

```
Widget
XmCreateXrtGaLEDIcon(
    Widget          parent,
    char *          name,
    Arg             *arglist,
    Cardinal        argcount
    )
```

parent is the parent; *name* is the newly created widget's name; *arglist* is a list, which can be `NULL`, of `XrtArgVal` resource/value pairs to provide to `XrtGaLEDIcon` upon creation; *argcount* is the number of arguments. It returns the newly created `XrtGaLEDIcon` widget or `NULL` if the widget could not be created.

XmIsXrtGaLEDIcon()

```
XmIsXrtGaLEDIcon(w)
```

Macro that returns `True` if `w` is an `XrtGaLEDIcon` widget or a subclass of an `XrtGaLEDIcon` widget.

5

The LEDNumerical Widget

Introduction ■ *The Value of an LED Numerical Widget*
Coloring the Widget and the Segments ■ *Dimensioning an LED Numerical widget*
Clock Displays ■ *Slanting the Numerals*
Callbacks ■ *Code Sample*
Widget Synopsis ■ *Resource Reference*
Procedures and Methods Reference

5.1 Introduction

The LEDNumerical widget provides a display that uses seven-segment cells to display numerals and some alphanumeric characters.

The widget is suitable for displaying scoreboards, digital time, and other simple numerical readouts.

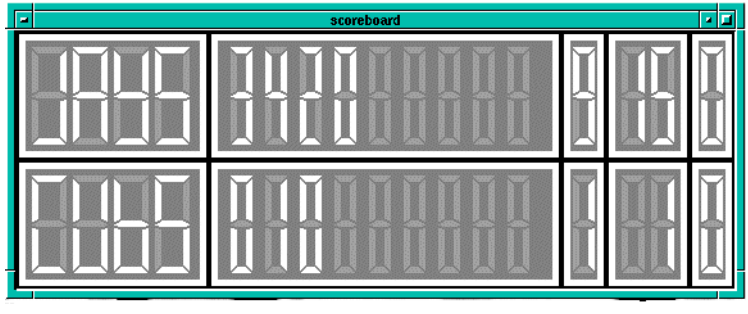


Figure 10 A baseball scoreboard formed from LEDNumerical widgets

5.2 The Value of an LED Numerical Widget

An LEDNumerical widget can have a numeric or an alphanumeric value. The choice is made through `XmNxrtGaValueType`, whose possible values are `XRTGA_VALUE_TYPE_NUMERIC` and `XRTGA_VALUE_TYPE_ALPHANUMERIC`. Choosing `XRTGA_VALUE_TYPE_ALPHANUMERIC` causes the display to do the best job that it can with the string given in `XmNxrtGaText`, taking into account the limitations of a 7-segment LED digit.

If the type is `XRTGA_VALUE_TYPE_NUMERIC`, you can choose to have a leading plus or minus sign by setting `XmNxrtGaLeadingPlusMinus` to `True`. Similarly, set `XmNxrtGaLeadingZeros` to `True` to fill the leading digits with zeros rather than leaving them blank.

You can control the number of digits to display using the `XmNxrtGaNumDigits` resource.

5.2.1 Setting Decimal Places

If `XmNxrtGaFixedDecimal` is `True` and the value type is numeric, use the value of the `XmNxrtGaDecimalPlaces` resource to control the number of places after the decimal point.

5.3 Coloring the Widget and the Segments

Segments have an *on* color and an *off* color. The resource names are `XmNxrtGaOnColor` and `XmNxrtGaOffColor`. If these resources are not set, the foreground and background colors respectively of the parent widget are used. The widget's background color is used fill any part of the widget not covered by an LED 7-segment cell.

As an alternative setting the *off* color, `XmNxrtGaColorMixPercentage` may be used to set the *off* color as a blend of the widget's background color and the segment's *on* color. This resource takes an integer value denoting the percentage of *on* color to mix with the background color. For example, a value of 20 means that the segment's *off* color is a blend of 20% *on* color and 80% background color.

5.4 Dimensioning an LED Numerical widget

The widget adjusts its size based on the width and height parameters it is given. Its height resource, `XmNheight`, sets the overall height of the widget. The width resource, `XmNwidth`, determines the width of the widget. This width should be large enough to contain all the digits specified in `XmNxrtGaNumDigits`. If not, the digit display is clipped by the widget. The width of a segment is controlled by `XmNxrtGaSegmentWidth`, and is independent of the other width setting.

Figure 11 shows the dimensions as they apply to a widget containing a single numeral. Note that the highlight thickness, border width, and horizontal spacing also play a part in determining the digit's width.

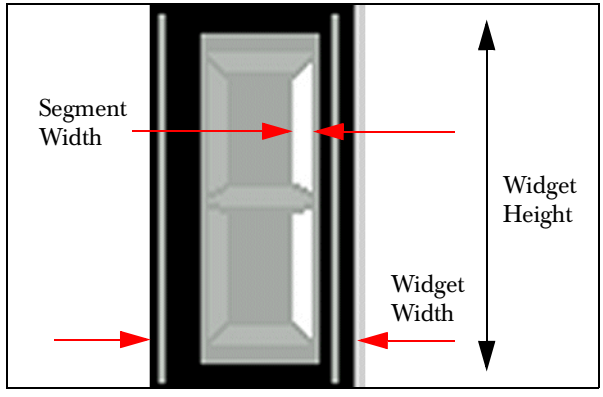


Figure 11 A sample LEDNumerical widget containing a single digit

Figure 12 shows the effect of increasing the overall width of the widget. The width of the numeral is the only variable parameter, so it increases to fill the additional space.

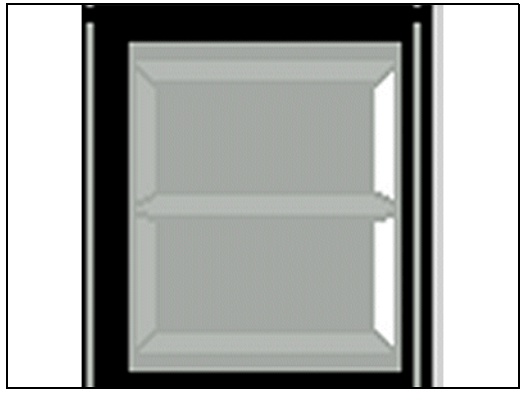


Figure 12 The effect of increasing the width of the widget, keeping other dimensions the same

The space between numerals is controlled by `XmNxrtGaSpacingHorizontal`. Figure 13 shows the effect of choosing a relatively large value for the horizontal spacing between numerals in the top widget containing the alphanumeric value "JAyS" while using a smaller value in the bottom "CUbS" widget.

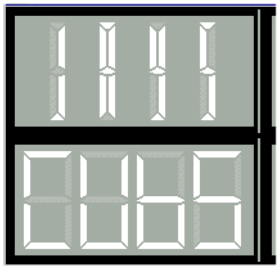


Figure 13 Two alphanumeric displays showing the effect of different horizontal spacings

Note that all dimension resources take parameters denoting their respective lengths in pixels.

5.5 Clock Displays

The tens digit on a clock has only two possible values, zero or one. Many LED displays are constructed so that the tens digit is a "two-segment" display, capable of showing the numeral "1" when on. A value of zero is denoted simply by turning off the display. To construct this type of display, set `XmNxrtGaLeadingOne` to `True`. This causes the first LED digit to be of the two-segment type.

5.6 Slanting the Numerals

Numerals are upright by default. They may be slanted either to the right or to the left with the `XmNxrtGaSlopeOffset` resource.

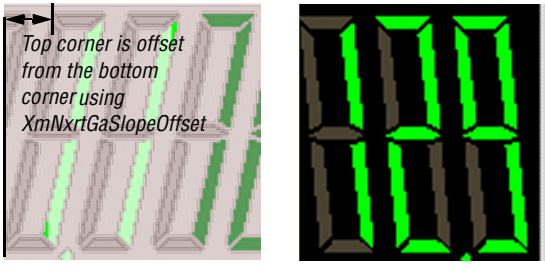


Figure 14 Setting a positive slope offset slants the numerals to the right

Figure 14 shows some italicized numerals. The figure was drawn by giving the slope offset parameter a value of 20 pixels. The numerals have been magnified to show the offset more clearly.The set of numerals with a darker background has been given a slope offset of -10.

5.7 Callbacks

The widget defines three callbacks, `XmNxrtGaExposeCallback`, `XmNxrtGaResizeCallback` and `XmNxrtGaValueChangedCallback`. See Chapter 22 for details.

5.8 Code Sample

To create an LEDNumerical widget within an application, simply set its resources as follows:

```
/* Create LED Numerical Widget*/
value_type = XRTGA_VALUE_TYPE_ALPHANUMERIC;
border = XRTGA_BORDER_PLAIN;
 jays =
    XtVaCreateManagedWidget("jays",
                            xmXrtGaLEDNumericalWidgetClass,form,
                            XmNheight,                          100,
                            XmNwidth,                           150,
                            XmNxrtGaValueType,            value_type,
                            XmNxrtGaText,                    "JAYS",
                            XmNxrtGaSegmentWidth,                5,
                            XmNxrtGaSpacingHorizontal,           5,
                            XmNxrtGaSpacingVertical,             5,
                            XmNxrtGaNumDigits,                   4,
                            XmNxrtGaBorder,                 border,
                            XmNshadowThickness,                  5,
                            XmNxrtGaColorMixPercentage,         20,
                            COLOR_RES(XmNborderColor,      "black"),
                            COLOR_RES(XmNxrtGaOnColor,     "white"),
                            XmNtopAttachment,         XmATTACH_FORM,
                            XmNleftAttachment,        XmATTACH_FORM,
                            COLOR_RES(XmNbackground,   "darkgreen"),
                             NULL);
```

The code snippet produces an LED numerical widget similar to that shown in the top left corner of Figure 10, above.

5.9 Widget Synopsis

Include File:	*<Xm/XrtGaLEDNumerical.h>*
Class Name:	XmXrtGaLEDNumerical
Class Hierarchy:	Core → XmPrimitive → XmXrtGaLEDNumerical
Class Pointer:	xmXrtGaLEDNumericalWidgetClass

For more information on the Motif widget hierarchy and where LEDNumerical fits into this hierarchy, see Figure 1, “XRT/gauge widgets’ and objects’ position within the Motif Class Hierarchy”.

5.10 Resource Reference

This section lists all of the resources for the Numeric LED 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** **inherited** **CSG**

Background color of the widget. If this resource is not set then the background color of the parent widget is used

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_3D_OUT** **CSG**

Border type of the widget. `XmNshadowThickness` is used for the border width of widgets.

XmNxrtGaColorMixPercentage **int** **30** **CSG**

This resource specifies how the *off* color is created by basing it on two other colors, `XmNbackground` and `XmNxrtGaOnColor`. A value 30 means that `XmNxrtGaOffColor`'s value is based on 30% of the *on_color* and 70% of the *background* color.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDecimalPlaces **int** **0** **CSG**

If `XmNxrtGaFixedDecimal` is `True` and the value type is numeric, use this resource value as the number of places after the decimal point.

XmNxrtGaDecimalPosition **Position** **0** **CSG**

The position in pixels that the decimal point should be placed below the baseline of the text or that colon symbols should off-centered with respect to the digits.

XmNxrtGaDecimalSize **Dimension** **5** **CSG**

The size in pixels of the dots in the decimal place symbols and in the colon symbols.

XmNxrtGaDoubleBuffer **Boolean** **True** **CSG**

Whether or not to double-buffer the drawing of the widget. Double buffering is a graphics technique that can reduce the "flickering" of an object when it is drawn or redrawn.

XmNxrtGaExposeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has repainted some portion of itself. A pointer to an `XrtGaCallbackStruct` is passed to each.

XmNxrtGaFixedDecimal **Boolean** **False** **CSG**

If `True` and the value type is numeric, use the `XmNxrtGaDecimalPlaces` resource to fix the number of decimal places.

XmNxrtGaLeadingOne **Boolean** **False** **CSG**

Display the numeral one as the leading digit. (turn on if present in the value or string, blank if not).

XmNxrtGaLeadingPlusMinus **Boolean** **False** **CSG**

Display a leading plus or minus sign, which is turned on if value type is numeric and non-zero or if the value type is alphanumeric and a leading "+" or "-" sign is present.

XmNxrtGaLeadingZeros **Boolean** **False** **CSG**

Display leading zeros in numeric mode. If `False`, those positions are blanked.

XmNxrtGaNumDigits **int** **5** **CSG**

The number of digits that are displayed, not including any leading plus, minus, or one-digit sign.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The widget's background image. If the value of this resource is `NULL` the background color is used.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are `XRTGA_IMAGE_LAYOUT_CENTERED` and `XRTGA_IMAGE_LAYOUT_TILED`.

XmNxrtGaOffColor **Pixel** ***dynamic**** ***CSG***

The off color of each digit. If this resource is not set, the background color of the parent widget is used.

* *The default value is dependent on* `XmNxrtGaColorMixPercentage`. *If* `XmNxrtGaColorMixPercentage` *is left at its default value, the off color is 30% of the on_color and 70% of the background color.*

XmNxrtGaOnColor **Pixel** ***inherited*** **CSG**

The on color of each digit. If this resource is not set, the foreground color of the parent widget is used.

XmNxrtGaRepaint **Boolean** **True** **CSG**

Whether or not this widget is to be redrawn when any of its resources are changed.

XmNxrtGaResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has resized. A pointer to an `XrtGaCallbackStruct` is passed to each.

XmNxrtGaSegmentSpacing **Dimension** **2** **CSG**

The spacing in pixels between adjacent LED segments.

XmNxrtGaSegmentWidth **Dimension** **5** **CSG**

The width in pixels of an LED segment.

XmNxrtGaSlopeOffset **int** **0** **CSG**

Specifies how many pixels the top left corner of an LED digit is offset horizontally from the bottom-left corner. This resource enables you to "italicize" the look of LED digits.

XmNxrtGaSpacingHorizontal **Dimension** **10** **CSG**

The spacing in pixels between digits.

XmNxrtGaSpacingVertical **Dimension** **2** **CSG**

The spacing in pixels above and below the LED digits.

XmNxrtGaText **String** **NULL** **CSG**

If the value type is alphanumeric, the text string pointed to by this resource is used to display the LED values. If this resource is NULL all digits are blank.

XmNxrtGaValue **double** **0.0** **CSG**

If the value type is numeric this resource is used to display the LED values.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

A list of callbacks called when the value changes. A pointer to an `XmNxrtGaValueChanged-CallbackSruct` is passed to each.

XmNxrtGaValueType **XrtGaValueType** **XRTGA_VALUE_TYPE_NUMERIC** **CSG**

Type of value used, which is either numeric or alphanumeric. If numeric (`XRTGA_VALUE_TYPE_NUMERIC`), the value is obtained from `XmNxrtGaValue`. If alphanumeric (`XRTGA_VALUE_TYPE_ALPHANUMERIC`), the value is obtained from `XmNxrtGaText`.

5.11 Procedures and Methods Reference

XmCreateXrtGaLEDNumerical()

Creates an LEDNumerical widget.

```
Widget
XmCreateXrtGaLEDNumerical(
    Widget          parent,
    char *          name,
    Arg             *arglist,
    Cardinal        argcount
    )
```

parent is the parent; *name* is the newly created widget's name; *arglist* is a list, which can be `NULL`, of `XrtArgVal` resource/value pairs to provide to `XrtGaLEDNumerical` upon creation; *argcount* is the number of arguments. It returns the newly created `XrtGaLEDNumerical` widget or `NULL` if the widget could not be created.

XmIsXrtGaLEDNumerical()

```
XmIsXrtGaLEDNumerical(w)
```

Macro that returns `True` if `w` is an `XrtGaLEDNumerical` widget or a subclass of an `XrtGaLEDNumerical` widget.

6

The Percent Widget

The Percent Widget ■ *Choosing a Scale*
Label Extents ■ *Placing a Label on an Item*
Notes on the Percent Widget's Resources ■ *Translations and Callbacks*
Widget Synopsis ■ *Resource Reference*
Procedures and Methods Reference

6.1 The Percent Widget

The Percent widget presents a view of how a group of related items combine to make up a whole, represented as 100%. The entity being represented is imagined to be made up of *items*. The items are quantified by a value parameter that in some meaningful way measures the magnitude of the item.

Sample Code

The following code snippet creates a Percent widget and sets some resources:

```
Widget scale;  /* The Percent widget's associated scale */
   percent =
       XtVaCreateManagedWidget("percent",
                               xmXrtGaPercentWidgetClass,         form,
                               XmNxrtGaScale,                    scale,
                               XmNxrtGaCalcMode,  XRTGA_CALC_MODE_PERCENT,
                               NULL);
```

A Percent widget with a circular scale has the appearance of a *pie chart*, a traditional way of representing the proportional parts of a whole. It's linear form resembles a *bar chart*, in which the width of each item is proportional to the value it represents.

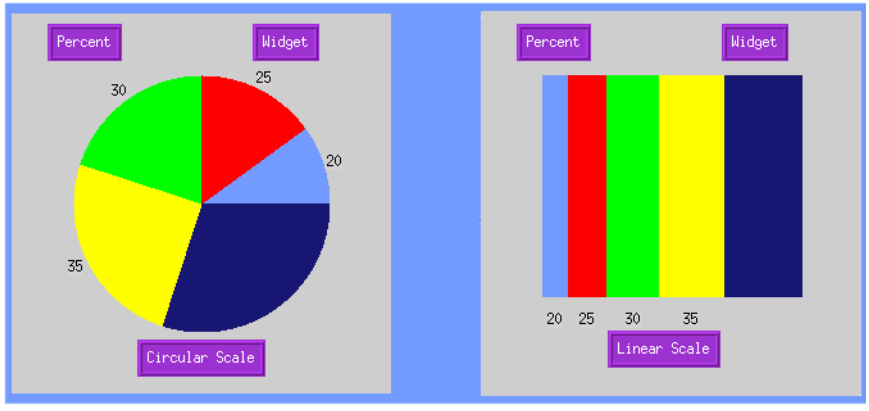


Figure 15 A Percent widget in its circular scale and linear scale modes

Figure 15 shows both forms the widget can take when it is in its *calculation-by-percent* mode. The darkest area on either scale is a visual representation of a "left over" part, which occurs when the sum of the items' values is less than the scale's total range, that is, the scale's maximum value minus its minimum value. The leftover part is stored in an internal item that can be accessed as part of the widget's `XmNxrtGaItemList`. Its value is the difference between the scale's *max* - *min* value and the sum of the values of the items.

There is an alternate way of performing the size calculation so that the leftover part is not displayed. In *calculation-by-ratio* mode, any leftover portion is ignored, and the items's values are summed, then each item is expressed as a fraction of that sum. Figure 16 illustrates the result.

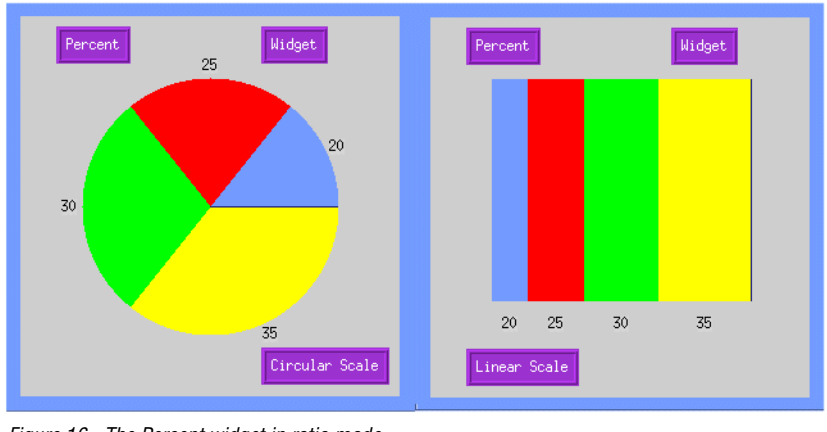


Figure 16 The Percent widget in ratio mode

An example may clarify how item values translate to labels on the scale. For the examples shown in Figure 15 and Figure 16, it is clear that the values shown beside the items are not percentages–they don't sum to 100 and the proportions are wrong. In fact, the item labeled 25 is 50% larger than the item labeled 20.

To illustrate how items set their values, both the linear scale and the circularly scale in the example have a minimum value 10 and a maximum value 110, for a difference of 100. The area of the entire scale is therefore 100 units. Each item calculates its share of the area using the formula

$$\frac{ItemValue - ScaleMin}{ScaleMax - ScaleMin}$$

The smallest item sets its value to 20. It calculates its share of the total area using the formula above, (20 - 10) / (110 - 10) = 0.1 = 10%. The largest item, whose value is 35, calculates its share of the total area as (35 - 10) / (110 - 10) = 25%. The share each item takes is then 10%, 15%, 20%, and 25%, for a total of 70%. There remains an "unused" area of 30%, which is shown as the unlabeled area in Figure 15. In the *calculation-by-ratio* mode, the area to be assigned to each item is based on the formula

$$\frac{ItemValue - ScaleMin}{\sum_{i}(ItemValue_i - ScaleMin)}$$

When the scale's minimum value is zero both formulas imply that an item's area is proportional to its value. In the case of *calculation-by-percent*, an additional area may be present to account for any deficit between the sum of the item values and the scale's *max - min* value. If this sum is greater than 100%, the items or the portion of an item that exceed the one hundred percent mark are not shown.

6.2 Choosing a Scale

The `XmNxrtGaScale` resource takes an object reference, which may be either a circular scale or a linear scale. If you create just one scale for the percent widget, it will automatically be registered as the scale to use. If you have created more than one scale, you can make a choice of the one to use by passing it in an `XtVaSetValues` call. The following example creates a circular scale and assigns it to the `XmNxrtGaScale` resource of the Percent widget:

```
Widget      percent;        /* A Percent widget */
Widget      circular_scale; /* A Circular Scale object for the
                               percent widget. */
Widget      linear_scale    /* For later use */

circular_scale =
    XtVaCreateWidget("circular_scale",
                     xmXrtGaCircularScaleObjectClass,       percent,
                     XmNxrtGaMin,                          DVAL(10.0),
                     XmNxrtGaMax,                         DVAL(110.0),
                     COLOR_RES(XmNbackground,       "midnightblue"),
                     XmNxrtGaStartAngle,                   DVAL(0.0),
                     XmNxrtGaStopAngle,                  DVAL(360.0),
```

```
                        NULL);
scale = circular_scale;
XtVaSetValues(percent,
                XmNxrtGaScale,  scale,
                NULL);
```

At another point in the application you may assign a linear scale to the `XmNxrtGaScale` resource. An identical call to `XtVaSetValues` causes the widget to display the linear scale.

The scale defines an overall size parameter for the widget whose value is the difference between its maximum value, set using `XmNxrtGaMax`, and its minimum value, set using `XmNxrtGaMin`.

XmNxrtGaItemList

An item list is updated as you add items to the Percent widget. Figure 17 shows the four items that have been added to the widget. In this case, the underlying circular scale has a start value of zero and a stop value of 100. Since the sum of the item's values is 90 and the mode is *calculation-by-percent*, the leftover item is displayed. These items are labeled on the diagram. All the items including the leftover item may be accessed via the item list.

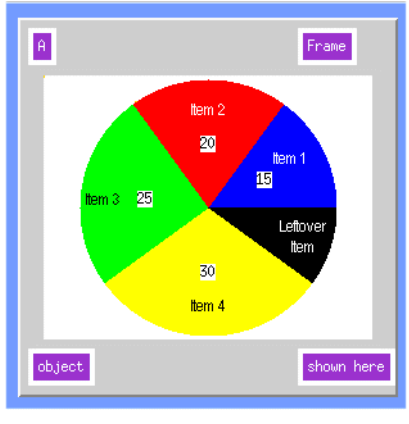


Figure 17 Percent widget showing a frame, items, and text

6.3 Label Extents

Each item may have a label whose position is controlled through `XmNxrtGaLabelExtent`:

```
item = XtVaCreateWidget(MtCreateName(name, "item"),
                        xmXrtGaItemObjectClass,          parent,
                        XmNxrtGaValue,                DVAL(20.0),
                        XmNxrtGaLabelExtent,           DVAL(0.5),
                        NULL);
```

Label extents begin at the center of a circular scale. An extent value of 1.0 positions the label at the circumference. The label extent for a linear scale begins at the left for a vertical scale and at the top for a horizontal scale. A label extent of 0.25 for a vertical scale places the label 25% of the way from the left side to the right.

Under certain conditions described next, the job of placing a label is given to the Item object, while the text is managed by the Text object.

6.4 Placing a Label on an Item

Use a Text object to place a label on an item. The Text object has an `XmNxrtGaValueObject` that associates it with an item as follows:

```
text =
     XtVaCreateWidget(MtCreateName(name, "text"),
                      xmXrtGaTextObjectClass,                  parent,
                      XmNxrtGaString,                            "%g",
                      XmNxrtGaIsShowing,                         True,
                      XmNxrtGaValueObject,                       item,
                      XmNxrtGaAllowWidgetPositioning,            True,
                      NULL);
```

Observe that `XmNxrtGaAllowWidgetPositioning` is `True`. This setting makes the item object responsible for setting the position. Notice that a value object is set to an Item object using

```
                      XmNxrtGaValueObject,                       item,
```

This informs the Text object to access the `item`'s `XmNxrtGaValue` resource for value information. The Text object then formats the value using a format specification such as

```
                      XmNxrtGaString,                            "%g",
```

to provide a convenient way of displaying an item's value. When the above-mentioned conditions are met, the Text object automatically takes the item's formatted value as its display text. Thus, specifying "`%g`" as the `XmNxrtGaString` allows the Text object to label an item with its value.

6.5 Notes on the Percent Widget's Resources

Placing a Frame

You can place a frame in a Percent widget by using the `XmNxrtGaFrame` resource. In Figure 17, the Percent widget has a size given by the gray rectangle. A white Frame object has been placed inside by setting margins as shown in the following code snippet:

```
frame =
    XtVaCreateWidget("frame",
                     xmXrtGaFrameObjectClass,                 percent,
                     XmNxrtGaBottomMargin,                         45,
                     XmNxrtGaTopMargin,                            45,
```

```
                    XmNxrtGaLeftMargin,                    20,
                    XmNxrtGaRightMargin,                   20,
                    COLOR_RES(XmNbackground,            "white"),
                    NULL);
```

XmNxrtGaTextList

Just as there is a list of Item objects, there is a list of text objects for the widget. Figure 17 shows four Text objects have been associated with the widget. These too are kept in a list and may be accessed using `XmNxrtGaTextList`.

Other resources

The Percent widget has a `XmNxrtGaImage` for placing a background image. It uses the `XmNxrtGaImageLayout` resource to set whether the image is centered or tiled. You can rearrange the order if list items using the `XmNxrtGaRenderList` resource, and specify that the render list should be used by setting the `XmNxrtGaUseRenderList` resource to True. Similarly, you can control when repaints occur by setting the `XmNxrtGaRepaint` Boolean resource. Setting it to `False` turns off repaints and setting it to `True` turns repainting on.

6.6 Translations and Callbacks

Translations for the Linear Gauge widget are:

```
CTRL+<Btn1Down>:   ZoomStart(Percent)
CTRL+<Btn1Motion>: ZoomUpdate(Percent)
CTRL+<Btn1Up>:     ZoomEnd(Percent)
SHIFT+<Btn1Down>:  TranslateStart(Percent)
SHIFT+<Btn1Motion>:TranslateUpdate(Percent)
SHIFT+<Btn1Up>:    TranslateEnd(Percent)
```

The callbacks for this widget are `XmNxrtGaExposeCallback`, `XmNxrtGaMapCallback` and `XmNxrtGaResizeCallback`. See Chapter 22 for details.

6.7 Widget Synopsis

Include Files:	*<Xm/XrtGaPercent.h>*
Class Name:	XmXrtGaPercentWidget
Class Hierarchy:	Core → XmPrimitive → XmXrtGaPercentWidget
Class Pointer:	xmXrtGaPercentWidgetClass

6.8 Resource Reference

This section lists all of the resources for the Percent 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the widget. If this resource is not set then the background color of the parent widget is used

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_3D_OUT** **CSG**

The border type of the widget.

XmNxrtGaCalcMode **XrtGaCalcMode** **XRTGA_CALC_MODE_PERCENT** **CSG**

Use this resource to determine the displayed percentage for each item. Valid values are `XRTGA_CALC_MODE_PERCENT` and `XRTGA_CALC_MODE_RATIO`. In percent mode, the percentage value of each item in the list is used by evaluating the formula (*value* - *min*) * 100 / (*max* - *min*). In ratio mode, the ratio of the percentage value to the total percentage value of every item on the list is used.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDoubleBuffer **Boolean** **True** **CSG**

Whether or not to double-buffer the drawing of the widget. Double buffering is a graphics technique that can reduce the "flickering" of an object when it is drawn or redrawn.

XmNxrtGaExposeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has repainted some portion of itself. A pointer to an `XrtGaCallbackStruct` is passed to each

XmNxrtGaFrame **XrtGaObject** **NULL** **CSG**

An `XrtGaFrame` object for the widget. The type of frame is controlled by the `XmNxrtGaShape` resource. By default no frame object is created for the parent widget. If a new frame is created for the widget it will replace the old one (if it exists) and the old one is kept in a list of frames maintained by the widget. If the current frame is destroyed, this resource is set to `NULL`.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The widget's background image. If this value is `NULL`, the background color is used.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are `XRTGA_IMAGE_LAYOUT_CENTERED` and `XRTGA_IMAGE_LAYOUT_TILED`.

XmNxrtGaItemList **XrtGaList** ***dynamic*** ***SG***

The list of items for the percent widget to display. Each new item object is automatically added to its parent's item list. This list is read-only and its contents should not manipulated by the user.

XmNxrtGaMapCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has had a `Map()` action registered. A pointer to an `XrtGaMapCallbackStruct` is passed to each map callback.

XmNxrtGaRenderList **XrtGaList** ***dynamic*** **G**

If the companion resource, `XmNxrtGaUseRenderList` is `True`, the widget will draw its object children in the order specified by the list stored in `XmNxrtGaRenderList`. When an object is created, it is automatically appended to the render list. Thus by default, the render list will have a list of object children in the order they were created. While this resource is read only, the user can get the pointer to the list and manipulate the list directly.

XmNxrtGaRepaint **Boolean** **True** **CSG**

Whether or not this widget is to be redrawn when any of its resources are changed.

XmNxrtGaResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has resized. A pointer to an `XrtGaCallbackStruct` is passed to each.

XmNxrtGaScale **XrtGaObject** **NULL** **SG**

The associated circular or linear scale that needs to be created to form the basis for displaying the items' values. If this value is `NULL`, the corresponding item display will not be drawn.

XmNxrtGaTextList **XrtGaList** **NULL** **SG**

A list of `XrtGaText` objects. If a text object needs to be associated with an item, the item should be passed to the `XmNxrtGaValueObject` resource of the text object. For bars, the text is centered in the item's area. For pies, the text is centered in the slice. For a circular scale, the item's `XmNxrtGaLabelExtent` resource is used to determine the text in relation to the scale's radius. For linear scales, the position of the label is related to the width or height of the scale.

XmNxrtGaUseRenderList **Boolean** **False** **G**

If the value of this resource is `True`, the widget draws its objects' children in the order specified by the list stored in `XmNxrtGaRenderList`. If this resource is `False`, the widget draws its objects' children in its default order, which should be sufficient for most situations. For example, the default order for the percent widget is frames, circular or linear scales, items and text objects associated with items, and finally text objects not associated with items.

6.9 Procedures and Methods Reference

XmCreateXrtGaPercent()

Creates a Percent widget.

```
Widget
XmCreateXrtGaPercent(
    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.

XmIsXrtGaPercent()

```
XmIsXrtGaPercent(w)
```

A macro that returns `True` if `w` is an `XrtGaPercent` widget or a subclass of an `XrtGaPercent` widget.

7

Center Object

XrtGaCenter ■ *Notes on the Center Object's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

7.1 XrtGaCenter

A center object is used to mark the position of the center of a circular scale in a Circular Gauge widget. It can be a colored circle or a user-supplied image. If an image is specified, the background will not be drawn. The image is not scaled to the radius of the center object.

A center object is associated with a circular scale, but it will not be drawn unless its `XmNxrtGaCircularScale` resource is set.

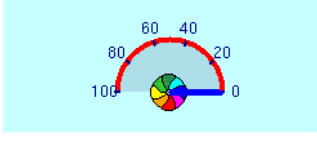


Figure 18 An image used as a center object

7.2 Notes on the Center Object's Resources

Setting the Center's color

Set the color of the Center object using the `XmNbackground` resource. The result is a circular disk of this color at the center of the scale. The size of the disk is controlled by the *radius* resource.

Associating a scale

Each circular scale object in a Circular Gauge may have one or more center objects. Specify on which circular scale the center is to be placed using the `XmNxrtGaCircularScale` resource.

Sizing the disk

Set the size of the disk using `XmNxrtGaRadius`. The value you pass it is a fraction based on the size of the radius of its associated circular scale. For instance, if you set a value of 0.2, the circular disk marking the center has a radius 20% as long as the radius of the circular scale.

Using an image as the visual

If you require anything fancier than a circular disk to mark the center of your scale, you can supply an image to `XmNxrtGaImage`. See the Sample Code for an example.

Controlling visibility

Use `XmNxrtGaIsShowing` to control whether the center object is visible.

7.3 Sample Code

Follow the first of these examples if you want to mark the center of circular scale with a colored disk and the second if you wish to supply an image.

```
Widget circulargauge; /* A Circular Gauge widget */
/*Create Center*/
center = XtVaCreateWidget("center",
                          xmXrtGaCenterObjectClass,    circulargauge,
                          XmNxrtGaCircularScale,              scale,
                          XmNxrtGaIsShowing,                   True,
                          XmNxrtGaRadius,                  DVAL(0.1),
                          COLOR_RES(XmNbackground,          "green"),
                          NULL);
```

If you wish to use an image, use:

```
char    *name;
Widget circulargauge;
name = "$XRTHOME/src/icons/disk_3D_blue.xpm";
/*Create Center*/
center = XtVaCreateWidget("center",
                          xmXrtGaCenterObjectClass,    circulargauge,
                          XmNxrtGaCircularScale,              scale,
                          XmNxrtGaIsShowing,                   True,
                          XtVaTypedArg,                XmNxrtGaImage,
                          XmRString,                           name,
                          strlen(name) + 1,
                          NULL);
```

An `XrtGaImage` is normally created through the image converter registered with each XRT/gauge widget. See Setting Images with XtConvertAndStore in Chapter 23 for details.

7.4 Resource Reference

This section lists all of the resources for the Center object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the object. If this resource is not set then the foreground color of the parent widget is used.

XmNxrtGaCircularScale **XrtGaObject** **NULL** **CSG**

The circular scale associated with the object. If this value is `NULL`, the object is not drawn.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* from the Xt error database. When False only resource conversion errors are printed.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The center's background image that is used if the type is image and the associated scale is linear. The image is drawn at the center of the associated circular scale.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the Center object.

XmNxrtGaRadius **double** **1.0** **CSG**

The outer extent of the center object. This value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale. For example, a value of 0.5 means the radius of the center object is half the size of the scale's radius.

7.5 Procedures and Methods Reference

XmCreateXrtGaCenter()

Standard Motif-style function to create an `XrtGaCenter` object:

```
Widget
    XmCreateXrtGaCenter (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaCenter` object to be created; *name* is the name to give the newly created `XrtGaCenter` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaCenter` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaCenter` object or `NULL` if the object could not be created.

XmIsXrtGaCenter()

```
XmIsXrtGaCenter(w)
```

Macro that returns `True` if `w` is an `XrtGaCenter` object or a subclass of an `XrtGaCenter` object.

8

Circular Scale Object

XrtGaCircularScale ■ *Angles In a Circular Scale*
Notes on Circular Scale Resources ■ *Sample Code*
Resource Reference ■ *Procedures and Methods Reference*

8.1 XrtGaCircularScale

A circular scale object may belong to a Circular Gauge Widget or a Circular Dial Widget. A Circular Dial keeps a reference to its circular scale using the `XmNxrtGaCircularScale` resource. In the case of the Circular Gauge, multiple circular scales may be attached. The Circular Gauge keeps a reference to them in the `XmNxrtGaCircularScaleList` resource.

8.2 Angles In a Circular Scale

The Circular Scale adopts an angular measurement convention that defines due east as the zero degree line. Angles increase in a counterclockwise direction, so that due north is 90°, due west is 180°, and due south is 270°.

Figure 19 illustrates the way that angular measurement is done in the Circular Scale, showing the *x* axis, *y* axis, and location of the zero degree line.

Once the start and stop angles of a circular scale have been chosen along with the maximum and minimum numerical values to be mapped on them, one additional parameter, *direction*, sets whether the start angle corresponds to the minimum value or the maximum value. The start angle is usually less than the stop angle but it is not a requirement. For example, if the start and stop angles are chosen as 90° and 270°, and the values being measured begin at 100 and end at 200, setting the direction to backward causes the value 200 to be located at 90°. The values decrease around the scale, ending at 270° where a value of 100 is located. Thus, while the coordinate

system for angular measurement never changes, the direction in which the scale's value parameter increases can be reversed.

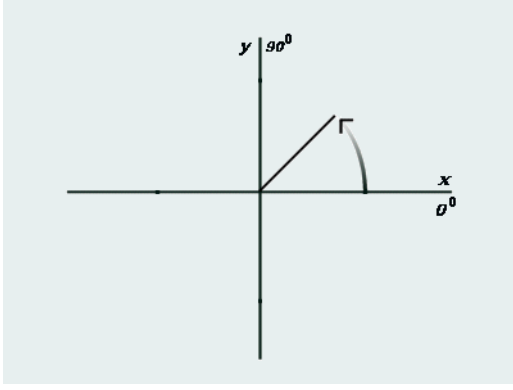


Figure 19 Diagram of a Circular Scale's reference plane.

8.3 Notes on Circular Scale Resources

The circular scale object's job is to hold the required numerical information for the construction of a Circular Gauge or a Circular Dial, but not any visual information except for the scale's background. Displaying the information is the responsibility of needles, tick marks, ranges, and so on. All of these are shown in Figure 20, along with the circular scale's resource values which place the scale within the widget, set the size of the circle, set the numerical range values, determine the part of the circle that is to be displayed, and which values the scale represents.

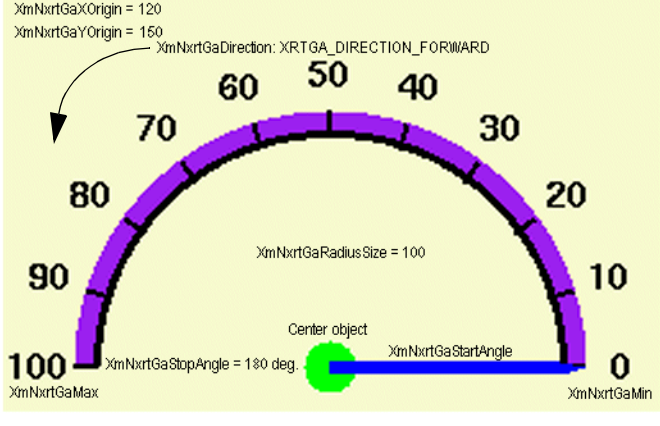


Figure 20 A circular scale showing representative values for its resources

A further Boolean resource, XmNxrtGaAllowTransform, controls whether the end user is allowed to use the mouse to move and resize the scale within the widget. If enabled, the XmNxrtGaTransformCallback points to a list of callbacks called during the transformation process. See Starting Translating and Zooming for details on the transform callback.

8.3.1 Specifying the Direction of Travel

The XmNxrtGaDirection resource controls the direction in which the scale's values increase. This resource can be set to any of the following:

XRTGA_DIRECTION_FORWARD	values increase counterclockwise
XRTGA_DIRECTION_BACKWARD	values increase clockwise

The default value for XmNxrtGaDirection is XRTGA_DIRECTION_FORWARD.

8.3.2 Specifying Start and Stop Angles

Set start and stop angles with XmNxrtGaStartAngle and XmNxrtGaStopAngle, for instance:

```
double min = 0.0;
double max = 100.0;
   /*Create Basic Scale*/
   scale = XtVaCreateWidget("scale",
                            xmXrtGaCircularScaleObjectClass,cgauge,
                            XmNxrtGaStartAngle,             DVAL(0.0),
                            XmNxrtGaStopAngle,            DVAL(180.0),
                            XmNxrtGaMin,                    DVAL(min),
                            XmNxrtGaMax,                    DVAL(max),
                            NULL);
```

8.3.3 Callbacks

This object supports a transform callback, XmNxrtGaTransformCallback. See Programming User Interaction for a discussion of these callbacks.

8.4 Sample Code

The following code sample creates a circular scale and sets specific resource:

```
   /*Create Basic Scale*/
   scale = XtVaCreateWidget("scale",
                       xmXrtGaCircularScaleObjectClass,         cgauge,
                       XmNxrtGaDirection,      XRTGA_DIRECTION_FORWARD,
                       XmNxrtGaMax,                         DVAL(100.0),
                       XmNxrtGaMin,                           DVAL(0.0),
                       XmNxrtGaRadiusSize,                          100,
                       XmNxrtGaStartAngle,                  DVAL(0.0),
                       XmNxrtGaStopAngle,                  DVAL(180.0),
                       XmNxrtGaXOrigin,                             0,
                       XmNxrtGaYOrigin,                             0,
                       XmNxrtGaRadiusSize,                        100,
```

```
                                    NULL);
    XtAddCallback(scale, XmNxrtGaTransformCallback, TransformCB, NULL);
```

8.5 Resource Reference

This section lists all of the resources for the circular scale object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the object. If this resource is not set then the foreground color of the parent widget is used.

XmNxrtGaAllowTransform **Boolean** **True** **CSG**

Whether or not the end user is allowed to zoom and translate the scale. Keyboard actions: *Ctrl <Btn1Drag>* : zoom, *Shift <Btn1Drag>* : translate.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False` only resource conversion errors are printed.

XmNxrtGaDirection **XrtGaDirection** **XRTGA_DIRECTION_FORWARD** **CSG**

The rotational direction from minimum to maximum on the scale. Possible values are `XRTGA_DIRECTION_FORWARD` and `XRTGA_DIRECTION_BACKWARD`. Forward is counterclockwise and backward is clockwise.

XmNxrtGaMax **double** **1.0** **CSG**

The maximum value the scale can have.

XmNxrtGaMin **double** **0.0** **CSG**

The minimum value the scale can have.

XmNxrtGaRadiusSize **Dimension** **dynamic** **CSG**

The scale's radius, which defaults to one-third of the minimum of the parent widget's width and height.

XmNxrtGaStartAngle **double** **0.0** **CSG**

The scale's start angle, in degrees.

XmNxrtGaStopAngle **double** **360.0** **CSG**

The scale's stop angle, in degrees.

XmNxrtGaTransformCallback	**XtCallbackList**	**NULL**	**CSG**

Called when the user zooms or moves the object.

XmNxrtGaXOrigin	**Position**	**dynamic**	**CSG**

The *x*-coordinate of the scale's origin. Defaults to the center of the parent widget.

XmNxrtGaYOrigin	**Position**	**dynamic**	**CSG**

The *y*-coordinate of the scale's origin. Defaults to the center of the parent widget.

8.6 Procedures and Methods Reference

This section lists the circular scale procedures and methods.

XmCreateXrtGaCircularScale()

Standard Motif-style function to create an `XrtGaCircularScale` object:

```
Widget
    XmCreateXrtGaCircularScale (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaCircularScale` object to be created; *name* is the name to give the newly created `XrtGaCircularScale` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaCircularScale` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaCircularScale` object or `NULL` if the object could not be created.

XmIsXrtGaCircularScale()

```
XmIsXrtGaCircularScale(w)
```

Macro that returns `True` if `w` is an `XrtGaCircularScale` object or a subclass of an `XrtGaCircularScale` object.

9

Counter Object

XrtGaCounter ■ Counter Resources
Callbacks and Translations ■ Sample Code
Widget Synopsis ■ Resource Reference
Procedures and Methods Reference

9.1 XrtGaCounter

A Counter object may be placed in a Circular Gauge or a Circular Dial.

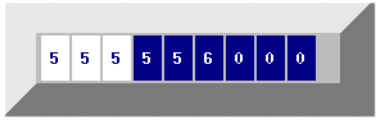


Figure 21 The Counter object

9.2 Counter Resources

9.2.1 Using the Counter Object as a Widget

Simply create a Circular Gauge or a Circular Dial and place a Counter object in it. If no other objects are created as children of the widget and the Counter object is sized appropriately, the widget simply acts as a container for the Counter.

9.2.2 Choosing a Border and a Background Color

You can place a border around a counter. There are many border styles to choose from. See this chapter's Resource Reference section for details.

The object's background color is visible around the edges of each numeral when either `XmNxrtGaSpacingHorizontal` or `XmNxrtGaSpacingVertical` are non-zero.

9.2.3 Choosing Colors

Besides the background color for the whole object, each numeral of the Counter object has a background color and a foreground color. The foreground color is used for the numeral itself while the background color is used for the rectangular shape holding the numeral.

Note: for digits to the right of the decimal place, background and foreground colors are reversed.

9.2.4 Choosing a Font

Fonts are usually set in a fallback array, for example:

```
static String fallbacks[] = {
    "*.counter.fontList: -*-times-medium-r-normal--24-140-*-*-*-*-*-*",
    NULL
    };
```

They may be set using `XtVaTypedArg` as well.

9.2.5 Using Images for the Counter's Background

You can choose a background image for the object. The image can be centered within the object, or it may be tiled.

9.2.6 Setting the Decimal Place

There is no decimal place icon in a Counter object. Digits before and after the decimal point interchange their foreground and background colors. If your application demands an explicit decimal icon, use the LEDNumerical widget.

9.2.7 Using Transitions

You can achieve the effect of having one digit roll over to its successor by setting `XmNxrtGaUseTransition` to `True`. This simulates the action of a mechanical counter in which the numerals zero through nine rotate smoothly into position. You control the rate of rotation with `XmNxrtGaTransitionRate`. Its value is the time in milliseconds between one frame and the next. The number of frames for a complete transition is set in `XmNxrtGaTransitionSteps`.

9.2.8 The Reset Button

You can give your users the option of resetting a counter by providing a reset button. It may be placed before or after the counter's digit display, and its shape may be rectangular, circular, or that of an image you provide.

You control the size of the reset button by setting its height, width, and offsets. The left and right offsets are controlled by `XmNxrtGaResetOffset` resource. The reset button is aligned vertically with the digits.

9.2.9 Setting a value for the Counter

Use the Counter's `XmNxrtGaValue` resource to give it the value that it should display. Make sure that you have specified a value in `XmNxrtGaNumDigits` that is large enough to hold the number of digits that your application needs to display.

9.3 Callbacks and Translations

Callbacks

Use the `XmNxrtGaValueChangedCallback` to react to a value change in the counter. See Chapter 22 for details.

Translations

When the reset button is showing, the widget automatically registers the *ResetCounter()* action with *<Btn1Down>* and *<Btn1Up>* events. The *ResetCounter()* action calls the Interaction Callback with reason `XRTGA_REASON_RESET_COUNTER_BEGIN` or `XRTGA_REASON_RESET_COUNTER_END` (depending on whether it is called from *<Btn1Down>*or *<Btn1Up>*) when the user presses the reset button of a counter object. The user must press *btn1down* and *btn1up* on the same counter for it to reset. Calling the callbacks enables the user to disable any corresponding needle/switch interaction when a counter button is pressed and enable it again when finished.

9.4 Sample Code

The following code sample creates a Counter object and many of its resources:

```
counter =
    XtVaCreateWidget("counter",
                     xmXrtGaCounterObjectClass,              cgauge,
                     XmNbackground,               COLOR_RES("grey"),
                     XmNxrtGaBorder,              XRTGA_BORDER_NONE,
                     XmNxrtGaBorderWidth,             (Dimension) 2,
                     XmNxrtGaDebug,                          True,
                     XmNxrtGaDecimalPlaces,                     2,
                     XmNxrtGaDigitBackgroundColor,  COLOR_RES("red"),
                     XmNxrtGaDigitForegroundColor, COLOR_RES("blue"),
                     XmNxrtGaDoubleBuffer,                   True,
                     XmNxrtGaImage,                          NULL,
                     XmNxrtGaImageLayout,   XRTGA_IMAGE_LAYOUT_TILED,
                     XmNxrtGaIsShowing,                      True,
                     XmNxrtGaNumDigits,                         7,
                     XmNxrtGaRepaint,                        True,
                     XmNxrtGaResetAlignment,     XRTGA_ALIGNMENT_END,
                     XmNxrtGaResetBackgroundColor,COLOR_RES("green"),
                     XmNxrtGaResetHeight,            (Dimension) 10,
                     XmNxrtGaResetImage,                     NULL,
                     XmNxrtGaResetOffset,             (Dimension) 2,
                     XmNxrtGaResetShape,          XRTGA_SHAPE_CIRCLE,
```

```
                                XmNxrtGaResetShow,                      True,
                                XmNxrtGaResetWidth,                (Dimension) 50,
                                XmNxrtGaSpacingHorizontal,         (Dimension) 15,
                                XmNxrtGaSpacingVertical,            (Dimension) 5,
                                XmNxrtGaTransitionRate,                    100,
                                XmNxrtGaTransitionSteps,                    10,
                                XmNxrtGaUseTransition,                  True,
                                XmNxrtGaValue,                     DVAL(12345.67),
                                NULL);

        XtAddCallback(counter, XmNxrtGaValueChangedCallback,
                      (XtCallbackProc)valueChangedCB, NULL);
```

9.5 Widget Synopsis

Include Files:	*<Xm/XrtGaCounter.h>*
Class Name:	XmXrtGaCounter
Class Hierarchy:	Core → XmPrimitive → XmXrtGaCounter
Class Pointer:	xmXrtGaCounterWidgetClass

9.6 Resource Reference

This section lists all of the resources for the Counter object, 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()*).

XmNbackground **Pixel** **inherited** **CSG**

Background color of the object. If this resource is not set then the background color of the parent widget is used

XmNfontList **XmFontList** **"fixed"** **CSG**

The font used for drawing any text for this object.

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

The height of the object.

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

The height of the object.

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

The horizontal location in pixels of the object's top left corner relative to its parent widget.

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

The vertical location in pixels of the object's top left corner relative to its parent widget.

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_3D_OUT** **CSG**

The border type of the object. Possible styles are:

`XRTGA_BORDER_NONE`	No border at all.
`XRTGA_BORDER_PLAIN`	The border is drawn in the foreground color.
`XRTGA_BORDER_3D_IN`	Area appears inset.
`XRTGA_BORDER_3D_OUT`	Area appears raised.
`XRTGA_BORDER_ETCHED_IN`	Double line, area appears inset.
`XRTGA_BORDER_ETCHED_OUT`	Double line, area appears raised.
`XRTGA_BORDER_FRAME_IN`	Shadow in at edge, area appears framed.
`XRTGA_BORDER_FRAME_OUT`	Shadow out at edge, area appears framed.
`XRTGA_BORDER_BEVEL`	The area appears framed.
`XRTGA_BORDER_SHADOW`	A shadow is drawn under area.

XmNxrtGaBorderWidth **Dimension** **2** **CSG**

Specifies the width of an object's border in pixels.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False` only resource conversion errors are printed.

XmNxrtGaDecimalPlaces **int** **0** **CSG**

The number of decimal places to use in the display. Negative numbers are treated as zero–no decimal places are displayed. Digits to the right of the decimal are in reverse video.

XmNxrtGaDigitBackgroundColor **Pixel** ***inherited*** **CSG**

The background color for each digit to the left of the decimal place, and the foreground color for each digit to the right of the decimal place. If this resource is not set, the background color of the parent widget is used.

XmNxrtGaDigitForegroundColor **Pixel** ***inherited*** **CSG**

The foreground color for each digit to the left of the decimal place, and the background color for each digit to the right of the decimal place. If this resource is not set, the foreground color of the parent widget is used.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The object's background image. If this value is `NULL`, the background color is used.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are `XRTGA_IMAGE_LAYOUT_CENTERED` and `XRTGA_IMAGE_LAYOUT_TILED`.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide a given object

XmNxrtGaNumDigits **int** **5** **CSG**

Number of digits to be displayed, not including any leading plus, minus, or one-digit signs.

XmNxrtGaResetAlignment **XrtGaAlignment** **XRTGA_ALIGNMENT_END** **CSG**

Specifies how the reset icon is aligned in the `XrtGaCounter` object. Possible values are `XRTGA_ALIGNMENT_BEGIN` and `XRTGA_ALIGNMENT_END`. Note that `XRTGA_ALIGNMENT_CENTER` is ignored.

XmNxrtGaResetBackgroundColor **Pixel** ***dynamic*** **CSG**

The background color for the reset button. If this resource is not set, the background color of the parent widget is used.

XmNxrtGaResetBackgroundColor **Pixel** ***inherited*** **CSG**

The background color of the reset button in a counter object.

XmNxrtGaResetImage **XrtGaResetImage** **NULL** **CSG**

The background image of the reset button. The image is drawn centered.

XmNxrtGaResetOffset **Dimension** **0** **CSG**

This offset specifies the horizontal offset distance for the *reset* icon. It is applied on the left and right side of the reset button.

XmNxrtGaResetShape **XrtGaResetShape** **XRTGA_SHAPE_RECTANGLE** **CSG**

The shape of the reset button. The values can be rectangular or circular, in which case the *on* and *off* colors are used to fill the specific shape, or image, in which case the image is used. The thermometer and tank shapes are not allowed for this resource. Possible values are:

`XRTGA_SHAPE_RECTANGLE`	Rectangular shape
`XRTGA_SHAPE_CIRCLE`	Circular shape.
`XRTGA_SHAPE_IMAGE`	Notifies the resource that an `XrtGaResetImage` specified by the `XmNxrtGaResetImage` resource is to be used to define the shape.

XmNxrtGaResetShow **Boolean** **False** **CSG**

If `True`, the reset button is drawn to the left of the digits. Clicking on it resets the counter value to zero.

XmNxrtGaResetWidth **Dimension** ***dynamic*** **CSG**

The width of the reset icon in the Counter object.

XmNxrtGaSpacingHorizontal **Dimension** **2** **CSG**

The spacing in pixels between digits.

XmNxrtGaSpacingVertical **Dimension** **5** **CSG**

The spacing in pixels above and below the digits.

XmNxrtGaTransitionRate **int** **10** **CSG**

The rate in milliseconds at which the counter increments between transisition steps.

XmNxrtGaTransitionSteps **int** **10** **CSG**

The number of steps between the current value and the next value.

XmNxrtGaUseTransition **Boolean** **False** **CSG**

If `True`, the transition rate and steps are used. Otherwise, the change between values in instataneous.

XmNxrtGaValue **double** **0.0** **CSG**

The value of the counter.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

The list of callbacks called when the value changes. A pointer to an `XmNxrtGaValueChanged-CallbackStruct` is passed to each.

9.7 Procedures and Methods Reference

This section lists the Counter procedures and methods.

XmCreateXrtGaCounter()

Creates a Counter object.

```
Widget
XmCreateXrtGaCounter(
    Widget      parent,
    char *      name,
    Arg         *arglist,
    Cardinal    argcount
    )
```

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

XmIsXrtGaCounter()

```
XmIsXrtGaCounter(w)
```

Macro that returns `True` if `w` is an `XrtGaCounter` object or a subclass of an `XrtGaCounter` object.

10

Frame Object

XrtGaFrame ■ *Notes on Setting the Frame Object's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

10.1 XrtGaFrame

Frames may be placed in Circular Gauge, Circular Dial, Linear Gauge, Linear Dial, and Percent widgets. One frame object per widget may be used to provide a colored background for the interior of the widget.

The top, left, right, and bottom margins are used to set the borders of the frame. In the widget's default rendering order, the frame is drawn immediately after the background.

An `XrtGaFrame` object's shape is controlled by the `XmNxrtGaShape` resource. Its type can be rectangular (the default), circular, or an image. If the type is circular, a frame in the shape of an ellipse is drawn that fits within the specified boundaries.

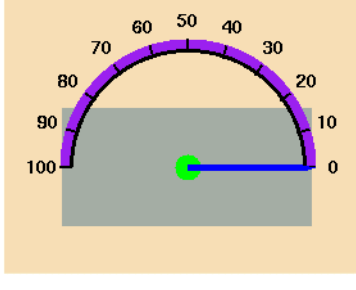


Figure 22 A gray frame set well inside its widget by adjusting margin settings

No frame object is created by default for the parent widget. If a new frame is created for the widget it will replace the old one (if it exists) and the old one is kept in a list of

frames maintained by the widget. If the active frame is destroyed, the widget's XmNxrtGaFrame resource is set to NULL.

10.2 Notes on Setting the Frame Object's Resources

Use XmNbackground to set the color of the frame.

Set XmNxrtGaDebug to True for additional debugging information.

To hide the frame, set XmNxrtGaIsShowing to False.

Set the frame type using XmNxrtGaShape. Visible frames can have a rectangular shape, a circular shape, or you can provide an image.

Control the placement of the frame with XmNxrtGaBottomMargin, XmNxrtGaLeftMargin, XmNxrtGaRightMargin, and XmNxrtGaTopMargin. Values for these margin resources are in pixels. Note that the margins begin at the value in pixels specified in the XmNxrtGaHighlightThickness resource. this is also where the widget's border begins.

10.3 Sample Code

The following code snippet demonstrates setting the frame's type, each margin's value, and the frame's color.

```
Widget cgauge; /* The parent widget - a circular gauge */
Widget cgauge_frame; /* The frame object */
/* Create frame for a Circular Gauge */
cgauge_frame =
    XtVaCreateWidget("cgauge_frame",
                     xmXrtGaFrameObjectClass,                cgauge,
                     XmNxrtGaFrameType,          XRTGA_SHAPE_CIRCLE,
                     XmNxrtGaIsShowing,                        True,
                     XmNxrtGaTopMargin,                           2,
                     XmNxrtGaLeftMargin,                          3,
                     XmNxrtGaRightMargin,                         4,
                     XmNxrtGaBottomMargin,                        5,
                     COLOR_RES(XmNbackground,               "white"),
                     NULL);
```

10.4 Resource Reference

This section lists all of the resources for the Frame object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the frame. If this resource is not set then the foreground color of the parent widget is used.

XmNxrtGaBottomMargin **int** **0** **CSG**

Specifies the bottom margin.

Frame margin resources do not start at the widget edge. Instead, they are referenced from the `XmNhighlightThickness`. The frame margin starts `XmNhighlightThickness` pixels in from the widget border. This is the same place that the border starts. Since the border is normally drawn over the frame, the margin must be greater than the border width for it to be seen.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The Frame objects's background image. The image is centered on the frame.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are `XRTGA_IMAGE_LAYOUT_CENTERED` and `XRTGA_IMAGE_LAYOUT_TILED`.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the frame.

XmNxrtGaLeftMargin **int** **0** **CSG**

Specifies the left margin.

Frame margin resources do not start at the widget edge. Instead, they are referenced from the `XmNhighlightThickness`. The frame margin starts `XmNhighlightThickness` pixels in from the widget border. This is the same place that the border starts. Since the border is normally drawn over the frame, the margin must be greater than the border width for it to be seen.

XmNxrtGaRightMargin **int** **0** **CSG**

Specifies the right margin.

Frame margin resources do not start at the widget edge. Instead, they are referenced from the `XmNhighlightThickness`. The frame margin starts `XmNhighlightThickness` pixels in from the widget border. This is the same place that the border starts. Since the border is normally drawn over the frame, the margin must be greater than the border width for it to be seen.

XmNxrtGaShape **XrtGaShape** **XRTGA_SHAPE_RECTANGLE** **CSG**

The shape of the frame. Valid values are `XRTGA_SHAPE_RECTANGLE` , `XRTGA_SHAPE_CIRCLE` , and `XRTGA_SHAPE_IMAGE` .

XmNxrtGaTopMargin **int** **0** **CSG**

Specifies the top margin. Any part of the object below this margin is drawn.

Frame margin resources do not start at the widget edge. Instead, they are referenced from the `XmNhighlightThickness`. The frame margin starts `XmNhighlightThickness` pixels in from the widget border. This is the same place that the border starts. Since the border is normally drawn over the frame, the margin must be greater than the border width for it to be seen.

10.5 Procedures and Methods Reference

XmCreateXrtGaFrame()

Standard Motif-style function to create an `XrtGaFrame` object:

```
Widget
    XmCreateXrtGaFrame (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaFrame` object to be created; *name* is the name to give the newly created `XrtGaFrame` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaFrame` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist.*

The function returns the newly created `XrtGaFrame` object or `NULL` if the object could not be created.

XmIsXrtGaFrame()

```
XmIsXrtGaFrame(w)
```

Macro that returns `True` if `w` is an `XrtGaFrame` object or a subclass of an `XrtGaFrame` object.

11

Icon Object

XrtGaIcon ■ *Notes on the Icon's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

11.1 XrtGaIcon

Icon objects are associated with the LEDIcon widget.

You form the visible part of the LEDIcon widget by adding as many *icons* as are necessary to show the progress of some measured quantity. Icons are granular markers whose job is to show the scale's current value. If the widget's mode is `XRTGA_DISPLAY_MODE_MAGNITUDE`, those icons whose values are less than the current value are shown in their *on* color, while all icons above the scale's value are shown in their *off* color. If the widget's mode is `XRTGA_DISPLAY_MODE_BINARY`, those icons whose values represent binary ones are shown in their *on* color, while all icons representing binary zeros are shown in their *off* color.

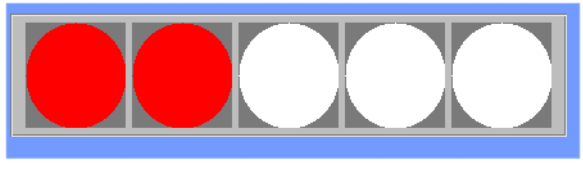


Figure 23 Circular icons whose off color is white

11.2 Notes on the Icon's Resources

11.2.1 Coloring the Icon

Three colors may be associated with an icon. `XmNbackground` is used to set the icon's background color. An icon's default rectangular shape fills all the allotted area and

leaves no room for the background color to be seen. The background color is visible when the icon's shape is circular, or when an image does not completely fill the icon.

Use `XmNxrtGaOnColor` and `XmNxrtGaOffColor` to control the icon's *on* and *off* colors.

11.2.2 Using Images in the Icon

Use `XmNxrtGaOnImage` and `XmNxrtGaOffImage` to control the icon's *on* and *off* images. A small image is centered in the icon, a large image is clipped.

11.2.3 Icon Shapes

The two possible shapes for an icon are rectangular and circular. The circle is drawn within the rectangular boundaries of the icon. If the icon's shape is rectangular, an ellipse is drawn instead. You can also use the `XmNxrtGaShape` resource to specify that an image is to be used rather than one of the built-in shapes.

Note that the thermometer and tank shapes are not available for icons.

11.3 Sample Code

```
Widget led; /* The parent LEDIcon widget */

/* Create an icon and add it to the icon list */
XtVaCreateWidget(icon_name,
                     xmXrtGaIconObjectClass,                  led,
                     COLOR_RES(XmNbackground,              "yellow",
                     COLOR_RES(XmNxrtGaOnColor,              "red",
                     COLOR_RES(XmNxrtGaOffColor,            "white",
                     XmNxrtGaShape,              XRTGA_SHAPE_CIRCLE,
                     NULL);
```

The icons in Figure 23 were created by setting resources as they are listed in this code example.

Alternatively, you can use `XmNxrtGaOnImage` and `XmNxrtGaOffImage` to have your icons display an *on* image and an *off* image.

11.4 Resource Reference

This section lists all of the resources for the Icon object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

The background color of the Icon object. If this resource is not set, the foreground color of the parent widget is used.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaOffColor **Pixel** ***inherited*** **CSG**

The color denoting the state of being deselected. The specified shape is filled with this color if the icon is off. If this resource is not set, the foreground color of the parent widget is used.

XmNxrtGaOnColor **Pixel** ***inherited*** **CSG**

The color denoting the state of being selected. The specified shape is filled with this color if the icon is on. If this resource is not set, the background color of the parent widget is used.

XmNxrtGaOnImage **XrtGaImage** **NULL** **CSG**

The icon's image when it is in the selected state. The image is drawn centered in the icon's space. The icon's size is determined by the parent. If the icon's size is too small for the image, the image is clipped to the icon's size.

XmNxrtGaOffImage **XrtGaImage** **NULL** **CSG**

The icon's image when it is in the deselected state. The image is drawn centered in the icon's space. The icon's size is determined by the parent. If the icon's size is too small for the image, the image is clipped to the icon's size.

XmNxrtGaShape **XrtGaShape** **XRTGA_SHAPE_RECTANGLE** **CSG**

The shape of the icon. The values can be `XRTGA_SHAPE_RECTANGLE` or `XRTGA_SHAPE_CIRCLE`, in which case the *on* and *off* colors are used to fill the specific shape, or `XRTGA_SHAPE_IMAGE`, in which case the image is used. The thermometer and tank range types are not allowed for this resource. The size of the icon is determined by the parent.

11.5 Procedures and Methods Reference

XmCreateXrtGaIcon()

Standard Motif-style function to create an `XrtGaIcon` object:

```
Widget
    XmCreateXrtGaIcon (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaIcon` object to be created; *name* is the name to give the newly created `XrtGaIcon` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaIcon` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaIcon` object or `NULL` if the object could not be created.

XmIsXrtGaIcon()

```
XmIsXrtGaIcon(w)
```

Macro that will return `True` if `w` is an `XrtGaIcon` object or a subclass of an `XrtGaIcon` object.

12

Indicator Object

XrtGaIndicator ■ *Notes on the Object's Resources*
Callbacks ■ *Sample Code*
Resource Reference ■ *Procedures and Methods Reference*

12.1 XrtGaIndicator

An indicator is a visual object used to provide an additional mark at the boundary between the *on* color (or image) and *off* color (or image) of a linear dial. It is also used in a linear gauge and may be placed anywhere between the *max* and *min* values of the linear scale with which it is associated. To be drawn, it requires that its associated scale is set. In the case of a linear dial, only one active indicator is possible and it needs to be associated with the widget's active linear scale.

There may be many indicators on a linear gauge. For a linear dial, an indicator is created for the parent widget by default. If a new indicator object is created and the widget's `XmNxrtGaIndicator` resource is at its default value, that is, pointing to the default indicator, this resource will automatically point to the new object and the default indicator will be destroyed. If the indicator to which this resource is pointing is destroyed, this resource is set to `NULL`.

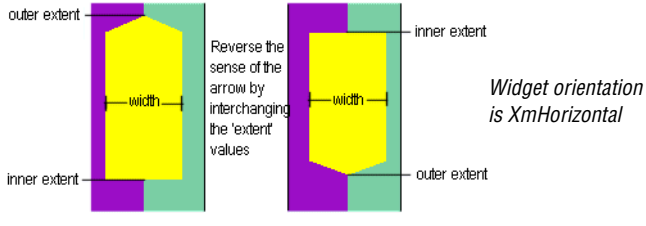


Figure 24 Light-colored indicators in the shape of a pointer

12.2 Notes on the Object's Resources

12.2.1 General Information

Use XmNxrtGaDebug for additional debug information.

Supply your own drawing function for a custom indicator and reference it with XmNxrtGaDrawFunc. See "Providing Your Own Draw Function" in the Advanced Topics chapter for details.

You can supply an image for an indicator. The XmNxrtGaImage resource is used to provide a reference for the image. Set XmNxrtGaImageRotate to True if you want the image to change its orientation when you change the orientation of the parent or when the outer extent becomes greater than the inner extent.

12.2.2 Indicator Extents

You control layout of the indicator by setting its XmNxrtGaInnerExtent and XmNxrtGaOuterExtent resources to control where its length dimension starts and stops, and by setting its XmNwidth resource to control its width. Both extents take parameters that are fractions based on the size of the associated linear scale. Refer to Figure 24 for an indicator on a horizontal linear scale whose inner extent is 0.8 and whose outer extent is 0.1.

12.2.3 Reversing the Sense of a Pointer or Triangle

If you wish to reverse the direction of pointer or triangle shapes, interchange the values in XmNxrtGaInnerExtent and XmNxrtGaOuterExtent. Interchanging the values produces a mirror image of the original shape.

12.3 Callbacks

The callback defined for this object is XmNxrtGaValueChangedCallback. It is a list of callbacks that are to be called when the indicator's value changes.A pointer to an XrtGaValueChangedCallbackStruct is passed to each. See Chapter 22 for details.

12.4 Sample Code

The following example places a blue indicator at the spot whose value is 20 on the scale with which it is associated.

```
Widget lgauge; /* The parent linear gauge */
Widget scale; /* A scale associated with the linear gauge*/
XrtGaIndicatorType indicator_type;

/*Create an indicator which will remain at the daily maximum value*/
   indicator_type = XRTGA_INDICATOR_TYPE_POINTER;
   max_indicator =
```

```
XtVaCreateWidget("max_indicator",
                 xmXrtGaIndicatorObjectClass,          lgauge,
                 XmNxrtGaLinearScale,                   scale,
                 XmNbackground,              COLOR_RES("grey"),
                 XmNwidth,                                 5,
                 XmNxrtGaValue,                    DVAL(20.0),
                 XmNxrtGaIndicatorType,          indicator_type,
                 XmNxrtGaInnerExtent,              DVAL(0.20),
                 XmNxrtGaOuterExtent,              DVAL(0.38),
                 COLOR_RES(XmNbackground,             "blue"),
                 XmNxrtGaIsShowing,                     True,
                 NULL);
```

12.5 Resource Reference

This section lists all of the resources for the Indicator object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

The background color of the object. If this resource is not set, the foreground color of the parent widget is used.

XmNwidth **Dimension** **0** **CSG**

The width in pixels of the widest part of the indicator. This resource is ignored if the type is circle or image.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDrawFunc **XrtGaDrawFunc** **NULL** **CSG**

Object drawing function. If this value is not `NULL`, it will be used to draw the indicator. The `XmNxrtGaIndicatorType` resource will be ignored. See Providing Your Own Draw Function on page 181 for details.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The image used for the indicator.

XmNxrtGaImageRotate **Boolean** **True** **CSG**

Specifies whether the image is to be rotated as the orientation changes, or as the values in `XmNxrtGaInnerExtent` and `XmNxrtGaInnerExtent` change.

XmNxrtGaInnerExtent **double** **0.0** **CSG**

The inner extent of the indicator object. For a horizontal scale, an inner extent of 0.2 means the range's inner extent is 0.2 times the scale's height away from the top of the scale. The scale's position determines the top left coordinates of the scale.

XmNxrtGaInteractionType **XrtGaInteractionType** **XRTGA_INTERACTION_TYPE_NONE** **CSG**

Specifies the type of mouse interaction with the indicator. The possible values are `XRTGA_INTERACTION_TYPE_NONE`, `XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE`, and `XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE`.

XmNxrtGaIndicatorType **XrtGaIndicatorType** **XRTGA_INDICATOR_TYPE_RECTANGLE** **CSG**

Type of indicator. The valid types are `XRTGA_INDICATOR_TYPE_RECTANGLE`, `XRTGA_INDICATOR_TYPE_CIRCLE`, `XRTGA_INDICATOR_TYPE_POINTER`, `XRTGA_INDICATOR_TYPE_TRIANGLE`, and `XRTGA_INDICATOR_TYPE_IMAGE`.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide a given object.

XmNxrtGaLinearScale **XrtGaObject** **NULL** **CSG**

The linear scale associated with the object. If this value is `NULL`, the indicator will not be drawn.

XmNxrtGaOuterExtent **double** **1.0** **CSG**

The outer extent of the indicator object. For a horizontal scale, an outer extent of 0.8 signifies that the range's outer extent is 0.8 times the scale's height away from the top of the scale. The scale's position determines the top left point of the scale.

XmNxrtGaValue **double** ***dynamic*** **CSG**

The indicator's current value, which should be between the associated scale's *min* and `max` values. If not, the value actually displayed will be confined to the scale's range, that is, a value greater than the maximum will be displayed at the *max* position and a value less than the minimum will be displayed at the *min* position.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

A list of callbacks that are to be called when the indicator's value changes. A pointer to an `XrtGaValueChangedCallbackStruct` is passed to each.

12.6 Procedures and Methods Reference

XmCreateXrtGaIndicator()

Standard Motif-style function to create an `XrtGaIndicator` object:

```
Widget
    XmCreateXrtGaIndicator (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaIndicator` object to be created; *name* is the name to give the newly created `XrtGaIndicator` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaIndicator` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaIndicator` object or `NULL` if the object could not be created.

XmIsXrtGaIndicator()

```
XmIsXrtGaIndicator(w)
```

Macro that will return True if w is an XrtGaIndicator object or a subclass of an XrtGaIndicator object.

13

Item Object

XrtGaItem ■ *Notes on the Object's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

13.1 XrtGaItem

A Percent widget is composed of *items*. Items are used to represent a set of quantifiable properties that may be grouped into a meaningful whole. Examples are the monetary values of stocks and bonds held in a portfolio, the composition by weight of elements in a chemical compound, or the numbers of employees in each department of an enterprise. Items have values whose sum must not be greater than the value set on the Percent widget itself.

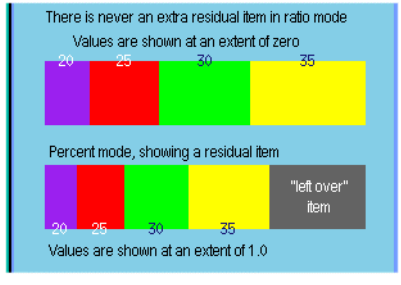


Figure 25 Colored items in a Percent widget, showing both ratio and percent modes

As shown in Figure 25, there is possibly an extra, "unnamed" icon in a Percent widget if the percent mode is chosen. This extra item occurs when the sum of the values in the named icons is less than the value determined by the *max* - *min* value in the associated circular or linear scale.

13.2 Notes on the Object's Resources

Using an image

Use the XmNxrtGaImage resource to set an image for the item. The image is centered within the item's area. This resource can only be used with linear scales.

Placing labels on items

Use a Text object to place a label on an item. The Text object has an XmNxrtGaValueObject that associates it with an item as follows:

```
text =
     XtVaCreateWidget(MtCreateName(name, "text"),
                      xmXrtGaTextObjectClass,                  parent,
                      XmNxrtGaString,                            "%g",
                      XmNxrtGaIsShowing,                         True,
                      XmNxrtGaValueObject,                       item,
                      XmNxrtGaAllowWidgetPositioning,            True,
                      NULL);
```

Observe that XmNxrtGaAllowWidgetPositioning is True. This setting makes the item object responsible for setting the position. Notice that a value object is set to an Item object using

```
                      XmNxrtGaValueObject,                       item,
```

This informs the Text object to access the item's XmNxrtGaValue resource for value information. The XmNxtrGaString resource then performs a format operation by being given a value such as

```
                      XmNxrtGaString,                            "%g",
```

to provide a convenient way of displaying an item's value. When the above-mentioned conditions are met, the Text object automatically takes the item's formatted value as its display text. Thus, specifying "%g" as the XmNxrtGaString allowsthe Text object to label an item with its value.

Label extents

Use the XmNxrtGaLabelExtent resource to position the label on the item. The resource's parameter is a fraction based on the associated circular or linear scale. For example, Figure 25 shows two linear scales. The the extent parameter on the upper one is zero, so the labels are positioned at the top of the scale. The the extent parameter on the lower one is 1.0, so the labels are positioned at the bottom of the scale.

For circular scales, label extents begin at the center and move out towards the circumference, where the extent is 1.0.

The value of the item

Set the item's value using the XmNxrtGaValue resource. When the Percent widget is in *calculation-by-ratio* mode, the item's values are summed and the total is used as the basis for assigning areas proportionally to each item for display. There is also a total value associated with the scale that the Percent widget is using. There is a leftover amount if the sum of all the item's values is less than the scale's total value. In this

case and when the Percent widget is in *calculation-by-percent* mode, the leftover item is assigned any leftover area (actually a portion of the pie in the case of a circular scale or a rectangle in the case of a linear scale). All items, including the leftover item, are displayed. The leftover item is displayed last.

Callbacks

Use the `XmNxrtGaValueChangedCallback` to react to changes in an item's value. It is a list of callbacks called when the item's value changes. A pointer to an `XmNxrtGaValueChangedCallbackStruct` is passed to each. See Chapter 22 for details.

13.3 Sample Code

The following code creates an item and assigns a value to it.

```
Widget percent; /* The parent */
Widget item; /* The Item object */

item = XtVaCreateWidget(MtCreateName(name,                    "item"),
                        xmXrtGaItemObjectClass,             percent,
                        XmNxrtGaValue,                  DVAL(value),
                        XmNxrtGaLabelExtent,              DVAL(0.5),
                        COLOR_RES(XmNbackground,             color),
                        NULL);

XtAddCallback(item /* Associate this item's value change with a CB */
              XmNxrtGaValueChangedCallback,
              itemCB, /* The callback function */
              (XtPointer)percent); /* The parent Percent widget */
```

13.4 Resource Reference

This section lists all of the resources for the Item object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** **NO_PIXEL_VALUE** **CSG**

Background color of the item. The item's background is set to `NO_PIXEL_VALUE` by default, which means the item's background is not drawn. If you set this resource to a valid pixel value, the item's background will *always* be drawn even if there is an image present. The object centers the image in the item and draws it on top of the background. Thus, if you want an image but no background, simply refrain from setting the background.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

Item objects's background image. The image is drawn centered on the Item object. This resource may be used only when the Percent widget is in linear mode.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

If `True`, the item is visible.

XmNxrtGaLabelExtent **double** **0.5** **CSG**

The extent of the text label for the item. If the associated scale is circular, this value is related to the `XmNxrtGaRadiusSize` value of the scale. For example, a value of 0.5 means the inner extent should begin one-half the distance between the scale's origin and the scale's circumference. For linear horizontal scales, an inner extent of 0.2 means the range's inner extent is 0.2 times the scale's height away from the top of the scale. The scale's position determines the top left point of the scale.

XmNxrtGaValue **double** **dynamic** **CSG**

The item's current value, which should be between the associated scale's *min* and `max` values. If not, the value actually displayed will be at one end of the scale's range, that is, a value greater than the maximum will be displayed at the *max* position and a value less than the minimum will be displayed at the *min* position.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

A list of callbacks called when the item's value changes. A pointer to an `XmNxrtGaValueChangedCallbackStruct` is passed to each.

13.5 Procedures and Methods Reference

This section lists the Item object procedures and methods.

XmCreateXrtGaItem()

Standard Motif-style function to create an `XrtGaItem` object:

```
Widget
    XmCreateXrtGaItem (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaItem` object to be created; *name* is the name to give the newly created `XrtGaItem` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaItem` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaItem` object or `NULL` if the object could not be created.

XmIsXrtGaItem()

```
XmIsXrtGaItem(w)
```

The macro that returns `True` if `w` is an `XrtGaItem` object or a subclass of an `XrtGaItem` object.

14

Linear Scale Object

XrtGaLinearScale ■ *Creating a Linear Scale*
User Interaction ■ *Sample Code*
Resource Reference ■ *Procedures and Methods Reference*

14.1 XrtGaLinearScale

The linear scale is the central object in a linear gauge or a linear dial. By setting its width and height, you control the size of the visible objects that will be associated with it. You can place tick marks on the scale, giving it a grid for making visual measurements, and you can place one or more ranges and indicators on the scale, providing additional visual information about particular values on the scale.

It's possible to employ a callback function to manipulate the indicator settings to reflect dynamically changing data, such as the maximum and minimum temperatures registered on a thermometer over a period of time. By assigning different colors to a pair of ranges, you can provide a clear visual indication of a particular value, such as the current temperature on the thermometer shown in Figure 26.

14.2 Creating a Linear Scale

Because the linear scale is the central object in a linear gauge or a linear dial, you will want to design its dimensional properties in conjunction with those of the widget itself. If you decide to place other objects, such as text areas, inside the widget, you will need to lay out all the objects and estimate their positions and dimensions. After your preliminary design is done, you can begin to configure the resources of the linear scale.

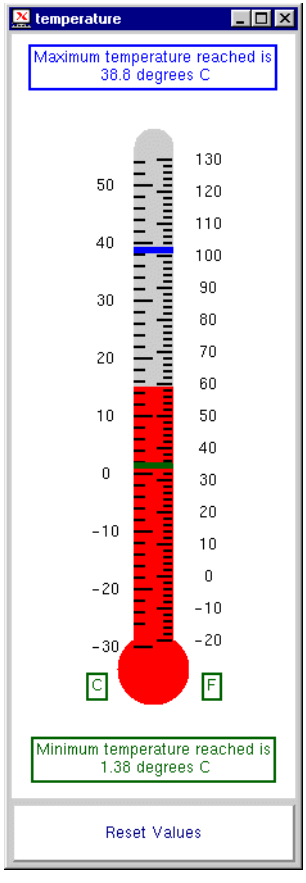


Figure 26 A linear gauge with Fahrenheit and Celsius scales

14.2.1 The Orientation and Direction of a Linear Scale

Even though a linear scale is not a visible object, its dimensions specify the location and the size of the scale. The visible objects, such as ranges, tick marks, and indicators, adjust their positions relative to the scale itself.

You position the scale by specifying its top left corner using `XmNx` and `XmNy`, and you set its rectangular size with `XmNheight` and `XmNwidth`.

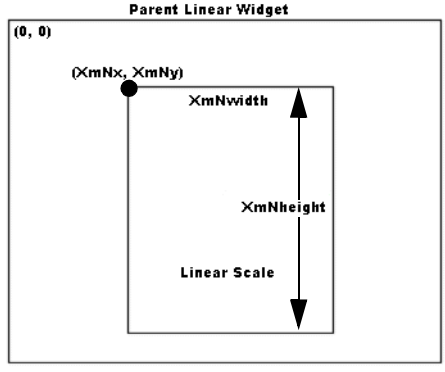


Figure 27 Setting dimensions on a linear scale

A linear scale may be aligned horizontally or vertically, and the direction in which a scale's values increase can be reversed. In its horizontal position, which is specified by setting `XmNxrtGaOrientation` to `XmHORIZONTAL`, values increase from left to right, or from bottom to top if the orientation is `XmVERTICAL`. To reverse the direction in which the scale's values increase, set `XmNxrtGaDirection` to `XRTGA_DIRECTION_BACKWARD`. When `XmNxrtGaDirection` is `XRTGA_DIRECTION_BACKWARD`, a horizontal scale's values increase from right to left, and a vertical scale's values increase from top to bottom.

14.3 User Interaction

Scales can be moved and resized by end users if you allow them to do so. This is done by setting the `XmNxrtGaAllowTransform` resource to `True`.

Callbacks

These transformations can also be controlled through the transform callback, `XmNxrtGaTransformCallback`. See Chapter 22, "Programming User Interaction", for details

14.4 Sample Code

Create a linear scale and set values on its resources by first specifying the widget with which the scale will be associated:

```
Widget         linear_gauge; /* The parent linear gauge Widget. /*
/*Create Basic Scale:*/
   scale = XtVaCreateWidget("scale",
                  xmXrtGaLinearScaleObjectClass,         linear_gauge,
                  XmNxrtGaMax,                             DVAL(60.0),
                  XmNxrtGaMin,                            DVAL(-40.0),
                  XmNxrtGaOrientation,                     XmVERTICAL,
```

```
XmNxrtGaDirection,          XRTGA_DIRECTION_FORWARD,
XmNwidth,                                       100,
XmNheight,                                      450,
XmNx,                                            50,
XmNy,                                            75,
NULL);
```

14.5 Resource Reference

This section lists all of the resources for the Linear scale object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** **NO_PIXEL_VALUE** **CSG**

Background color of the scale. If this is not set, no background is drawn for the scale.

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

The height of the linear scale, which defaults to the height of the parent widget.

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

The width of the linear scale, which defaults to the width of the parent widget.

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

The *x*-coordinate of the top left corner of the linear scale.

XmNxrtGaAllowTransform **Boolean** **True** **CSG**

Whether or not the end user is allowed to zoom and translate the scale. Keyboard actions: *<Btn1Drag>* – scale, *Shift<Btn1Drag>* – translate.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDirection **XrtGaDirection** **XRTGA_DIRECTION_FORWARD** **CSG**

The direction from which *min* goes to *max*. Possible values are `XRTGA_DIRECTION_FORWARD` and `XRTGA_DIRECTION_BACKWARD`. For linear objects, forward means left to right or bottom to top, depending on the orientation.

XmNxrtGaMax **double** **1.0** **CSG**

The scale's maximum value.

XmNxrtGaMin **double** **0.0** **CSG**

The scale's minimum value.

XmNxrtGaOrientation **unsigned char** **XmHORIZONTAL** **CSG**

The object's orientation. Possible orientations are `XmHORIZONTAL` and `XmVERTICAL`.

XmNxrtGaTransformCallback **XtCallbackList** **NULL** **CSG**

A list of callbacks that are called when the user zooms or moves the scale. A pointer to `XmNxrtGaTransformCallbackStruct` is passed to each.

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

The *y*-coordinate of the top left corner of the linear scale.

14.6 Procedures and Methods Reference

This section lists the Linear scale procedures and methods.

XmCreateXrtGaLinearScale()

Standard Motif-style function to create an `XrtGaLinearScale` object:

```
Widget
    XmCreateXrtGaLinearScale (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaLinearScale` object to be created; *name* is the name to give the newly created `XrtGaLinearScale` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaLinearScale` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaLinearScale` object or `NULL` if the object could not be created.

XmIsXrtGaLinearScale()

```
XmIsXrtGaLinearScale(w)
```

Macro that returns `True` if `w` is an `XrtGaLinearScale` object or a subclass of an `XrtGaLinearScale` object.

15

Marker Object

XrtGaMarker ■ Notes on the Object's Attributes
Sample Code ■ Resource Reference
Procedures and Methods Reference

15.1 XrtGaMarker

A Marker is an object used to highlight the Switch object's current position on a Circular Dial. Figure 28 shows a triangular marker placed at the 270° position on a circular dial. The marker follows the value around the scale. If the scale is in continuous mode, the marker moves smoothly. If the scale is in discrete mode, the marker assumes only those positions specified in the discrete states.

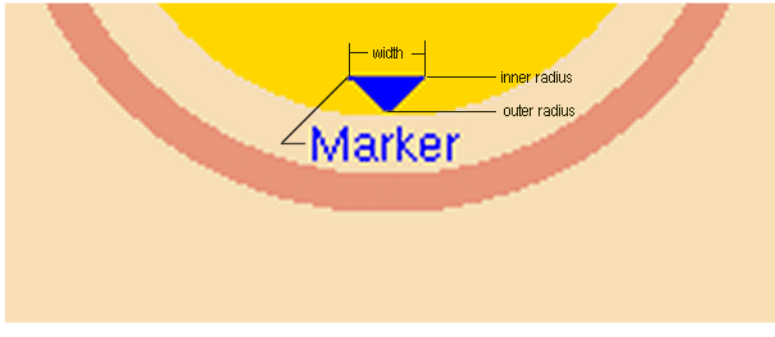


Figure 28 A triangular marker

A marker's dimensions are defined by specifying its inner radius and its outer radius, which are given as ratios of the scale's radius, and its width, which is given in pixels (when the marker is an arrow or a triangle). If the inner radius is greater than the outer radius, the radius values effectively switch.

The resources corresponding to the above-mentioned parameters are `XmNwidth`, `XmNxrtGaInnerRadius`, and `XmNxrtGaInnerRadius`.

Using the inner radius, outer radius, and width, a marker can be sized and positioned quite flexibly, as shown in Figure 29.

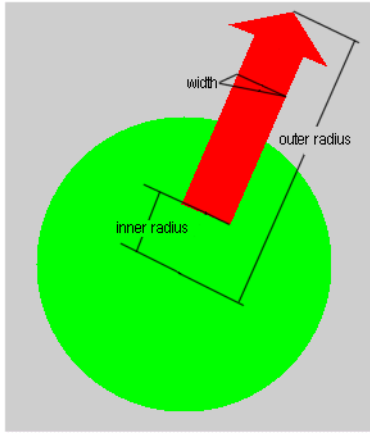


Figure 29 An arrow marker

15.2 Notes on the Object's Attributes

Set `XmNxrtGaDebug` to `True` for extra debugging information.

Provide a reference to a drawing function in `XmNxrtGaDrawFunc` if you wish to draw your own marker. See Providing Your Own Draw Function on page 181 for details.

Use `XmNxrtGaImage` to specify an image for the marker and if you want the image to rotate as the marker moves around the scale, set `XmNxrtGaImageRotate` to `True`.

You can choose to show or hide a marker by setting the value of the `XmNxrtGaIsShowing` resource.

15.3 Sample Code

The following code snippet sets most of the resource values for a Marker object:

```
/* Create triangular marker for a circular dial */
ld_marker =
    XtVaCreateWidget("ld_marker",
                     xmXrtGaMarkerObjectClass,          light_dial,
                     XmNxrtGaCircularScale,               ld_scale,
                     XmNwidth,                                  10,
                     COLOR_RES(XmNbackground,                "red"),
                     XmNxrtGaInnerRadius,                DVAL(1.5),
                     XmNxrtGaOuterRadius,               DVAL(0.55),
                     XmNxrtGaMarkerType, XRTGA_MARKER_TYPE_TRIANGLE,
                     XmNxrtGaIsShowing,                       True,
                     NULL);
```

15.4 Resource Reference

This section lists all of the resources for the Marker object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

The background color of the Marker object. If this resource is not set, the foreground color of the parent widget is used.

XmNwidth **Dimension** **4** **CSG**

The width in pixels of the widest part of the marker. This resource is ignored if the type is circle or image.

XmNxrtGaCircularScale **XrtGaObject** **NULL** **CSG**

The circular scale associated with this object. If this value is `NULL`, the object will not be drawn.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDrawFunc **XrtGaDrawFunc** **NULL** **CSG**

Object drawing function. If this value is not `NULL`, it will be used to draw the object. The `XmNxrtGaMarkerType` resource will be ignored.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The marker's background image (used if the type is image). The image that is drawn will be centered between the inner and outer radius. Images are not rotated depending on the value of the parent's switch object.

XmNxrtGaImageRotate **Boolean** **True** **CSG**

Specifies whether the image is to be rotated as the angle changes.

XmNxrtGaInnerRadius **double** **0.85** **CSG**

The inner extent of the marker object. This value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale. For example, a value of 0.5 means that the inner radius of the marker object is half-way between the scale's origin and its circumference.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the marker.

XmNxrtGaMarkerType **XrtGaMarkerType** **XRTGA_MARKER_TYPE_CIRCLE** **CSG**

The type of marker. The valid values are circle, triangle, arrow, and image.

XmNxrtGaOuterRadius	double	0.95	CSG

The outer extent of the marker object. This value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale. For example, a value of 1.5 means the outer extent of the marker object is located one-and-one-half times the size of the scale's radius away from the scale's origin.

15.5 Procedures and Methods Reference

XmCreateXrtGaMarker()

Standard Motif-style function to create an `XrtGaMarker` object:

```
Widget
    XmCreateXrtGaMarker (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaMarker` object to be created; *name* is the name to give the newly created `XrtGaMarker` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaMarker` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaMarker` object or `NULL` if the object could not be created.

XmIsXrtGaMarker()

```
XmIsXrtGaMarker(w)
```

Macro that returns `True` if `w` is an `XrtGaMarker` object or a subclass of an `XrtGaMarker` object.

16

Needle Object

XmNxrtGaNeedle ■ *Code Sample*
Resource Reference ■ *Procedures and Methods Reference*

16.1 XmNxrtGaNeedle

Needle objects are the visible pointers on a Circular Scale object in a Circular Gauge widget. The Circular Gauge widget allows more than one needle to be displayed on the same circular scale. Each needle has its own value and this value determines where on the scale it is positioned.

A needle can be positioned by either clicking or dragging on the scale. The Circular Gauge has an interaction callback that allows it to respond to mouse actions. When a needle's value changes a value changed callback is called to allow the user to disable or limit the change.

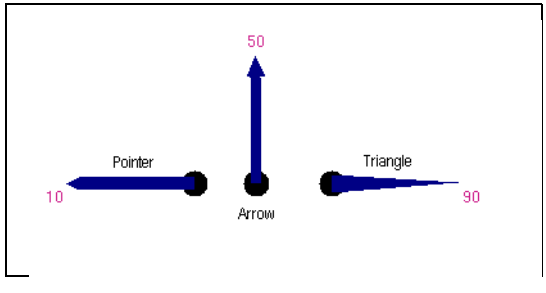


Figure 30 The three built-in needle shapes

16.2 A Note on the Needle Object's Resources

Needle shapes

A needle has three possible built-shapes: arrow, pointer, or triangle, as shown in Figure 30. The needle's length is set using the `XmNxrtGaLength` resource and is based on the associated circular scale. A needle begins at the center of the circular scale and extends outwards. For example, if its `XmNxrtGaLength` resource is set to 1.0, the needle's tip lies on the circumference of the associated circular scale. A needle's width is set using the `XmNwidth` resource.

It is possible to provide your own drawing routine if you require a needle with a custom shape. See Providing Your Own Draw Function on page 181 for details.

Needles may be sized however you wish, but if the length specification causes the needle to be drawn outside the widget, the needle will be clipped by the widget.

Coloring a Needle object

A needle's color is set using the `XmNbackground` resource.

Controlling a Needle's visibility

You can show or hide a needle by setting its `XmNxrtGaIsShowing` resource to `True` or `False` respectively.

Positioning a needle with the mouse

Use the `XmNxrtGaInteractionType` resource to determine how the needle responds to mouse clicks within the Circular Gauge widget to stop the needle from responding to the mouse. Possible values are *snap-to-mouse*, *relative-to-mouse*, or *none*.

16.3 Code Sample

You can completely configure a needle at the time it is created. The following code sample shows how:

```
/*Create Needle*/
interaction_type = XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE;
needle_type = XRTGA_NEEDLE_TYPE_POINTER;
needle = XtVaCreateWidget("needle",
                          COLOR_RES(XmNbackground,       "navyblue"),
                          XmNwidth,                              5,
                          XmNxrtGaCircularScale,              scale,
                          xmXrtGaNeedleObjectClass,          cgauge,
                          XmNxrtGaInteractionType,  interaction_type,
                          XmNxrtGaIsShowing,                   True,
                          XmNxrtGaLength,                       1.0,
                          XmNxrtGaNeedleType,           needle_type,
                          XmNxrtGaValue,                         33,
                          NULL);
```

Callbacks

The callback registered for this object is `XmNxrtGaValueChangedCallback`, which is called whenever the needle's value changes. It is a list of callbacks called when the

value changes. A pointer to an `XmNxetGaValueChangedCallbackStruct` is passed to each. See Chapter 22 for details.

16.4 Resource Reference

This section lists all of the resources for the needle object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the Needle object. If this resource is not set then the foreground color of the parent widget is used.

XmNwidth **Dimension** **0** **CSG**

The width in pixels of the widest part of the object. If this value is 0, draw a line of width 0 (the thinnest line possible) no matter what the type.

XmNxrtGaCircularScale **XrtGaObject** **NULL** **CSG**

The circular scale to which the needle belongs. If this value is `NULL`, the object will not be drawn.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDrawFunc **XrtGaDrawFunc** **NULL** **CSG**

The drawing function for the object. If this value is not `NULL`, it is used to draw the object, ignoring the `XmNxrtGaNeedleType`.

XmNxrtGaInteractionType **XrtGaInteractionType** **XRTGA_INTERACTION_TYPE_NONE** **CSG**

The type of mouse interaction with the object. The possible values are `XRTGA_INTERACTION_TYPE_NONE`, `XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE`, and `XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE`.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the needle.

XmNxrtGaLength **double** **1.0** **CSG**

This value determines the length of the needle. Its value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale. For example, a value of 0.5 means the needle should be 1/2 the size of the scale's radius. The needle is anchored at the scale's origin.

XmNxrtGaNeedleType **XrtGaNeedleType** **XRTGA_NEEDLE_TYPE_POINTER** **CSG**

The type of the needle. The valid types are `XRTGA_NEEDLE_TYPE_POINTER`, `XRTGA_NEEDLE_TYPE_TRIANGLE`, and `XRTGA_NEEDLE_TYPE_ARROW`.

XmNxrtGaValue **double** ***dynamic*** **CSG**

The needle's current value. This should be between the associated scale's *min* and *max* value. If not, the value actually displayed will be confined to the scale's range that is, a value greater than the maximum allowable is displayed at the *max* position, and a value less than the minimum allowable is displayed at the *min* position.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

A list of callbacks called when the value changes. A pointer to an `XmNxetGaValueChangedCallbackStruct` is passed to each.

16.5 Procedures and Methods Reference

This section lists the circular scale procedures and methods.

XmCreateXrtGaNeedle()

Standard Motif-style function to create an `XrtGaNeedle` object:

```
Widget
    XmCreateXrtGaNeedle (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaNeedle` object to be created; *name* is the name to give the newly created `XrtGaNeedle` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaNeedle` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaNeedle` object or `NULL` if the object could not be created.

XmIsXrtGaNeedle()

```
XmIsXrtGaNeedle(w)
```

Macro that returns `True` if `w` is an `XrtGaNeedle` object or a subclass of an `XrtGaNeedle` object.

17

Range Object

XrtGaRange ■ *Notes on the Range Object's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

17.1 XrtGaRange

A Range object may be created for these widgets: Circular Gauge, Circular Dial, and Linear Gauge. It is a visible object that may be used to mark the entire span or any sub-portion of a scale. A range can be specified for either a circular scale or for a linear scale.

Figure 31 shows two range objects on a circular scale. The thinner one spans the entire scale and appears as a semi-circle. The thicker one spans the region tick marks 20 and 80. The diagram shows how a start value, stop value, inner extent, and outer extent determine the shape of a range.

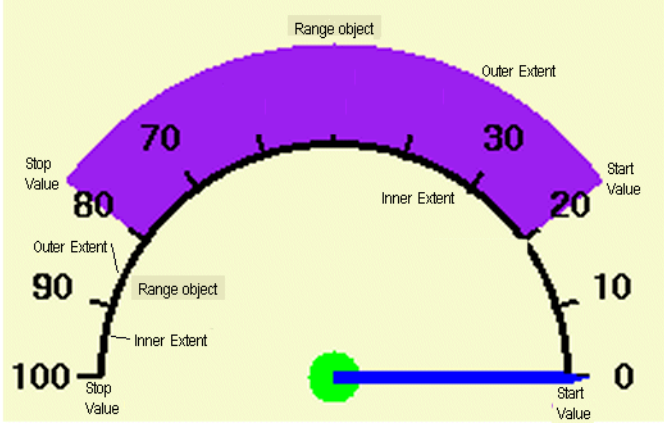


Figure 31 Two Range objects for a Circular Scale object

A Range object on a linear scale behaves similarly. Figure 32 shows a Linear Gauge widget. The lighter color shows the size of the widget while the darker color shows the size of the linear scale associated with the widget because the Range object has had its inner extent set to zero and its outer extent set to 1.0. Default settings for start and stop values are used resulting in a range that is the same size as the linear scale.

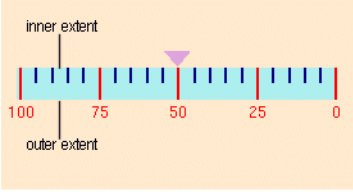


Figure 32 A range object that is the same size as its associated linear scale

17.2 Notes on the Range Object's Resources

The `XmNbackground` is used to set the color of the range.

Associate a Range with a Circular Scale

A Range object has to be associated with a parent scale by setting its `XmNxrtGaScale` resource.

A range spans a region of its associated scale, but to be visible it must have a thickness as well as a span. Resources `XmNxrtGaInnerExtent` and `XmNxrtGaOuterExtent` control the inner and outer limits of its thickness. Values for these resources are given as ratios based on the size of its associated scale. For example, an inner extent of 0.75 and an outer extent of 1.25 means that the thickness is half the width (or height, depending on the orientation) of an associated linear scale, or half the radius of an associated circular scale. If the range is associated with a linear scale, measurements are taken from the left side of a vertical scale or from the top of a horizontal scale.

Figure 31 shows two Range objects on a circular scale. The range that goes from 0 to 100 has an inner radius of 1.0 and an outer radius of 1.0, therefore it lies along the circumference of the circular scale[1]. The other range has an inner radius of 1.0 and an outer radius of 1.2, so it extends outwards from the circumference a distance equal to 20% of the radius.

1. When the inner radius and outer radius are the same, the thinnest possible line is drawn.

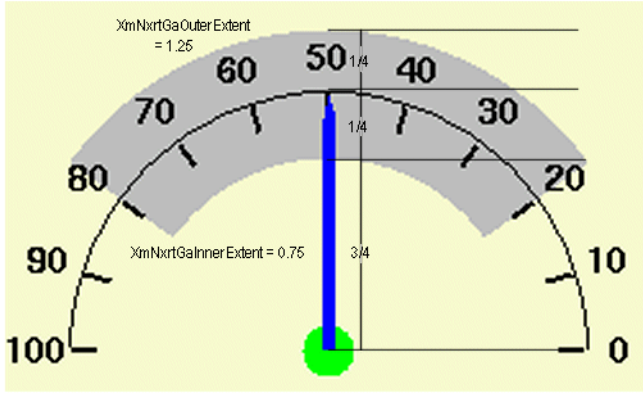


Figure 33 How extents relate to the size of the associated scale

Use `XmNxrtGaIsShowing` to control when the range is visible as the application runs.

Shapes for a Range object

On a circular scale, ranges are always figures bounded by two radial straight lines and two circular arcs, as shown in Figure 33, but there are choices for the shape of a range that is associated with a linear scale. The `XmNxrtGaShape` resource controls whether the shape is rectangular (the default) or circular, or has the appearance of a thermometer or of a rounded tank. As well, an image can be specified for linear scales only using `XmNxrtGaImage` and `XmNxrtGaImageLayout`.

Resources `XmNxrtGaStartValue`, `XmNxrtGaStopValue`, and `XmNxrtGaOffset` (described next) set the span of the scale's values that are marked by the range object.

Extending past the scale

Use an offset to shift a range beyond the limits imposed by the circular scale associated with the Circular Gauge widget.

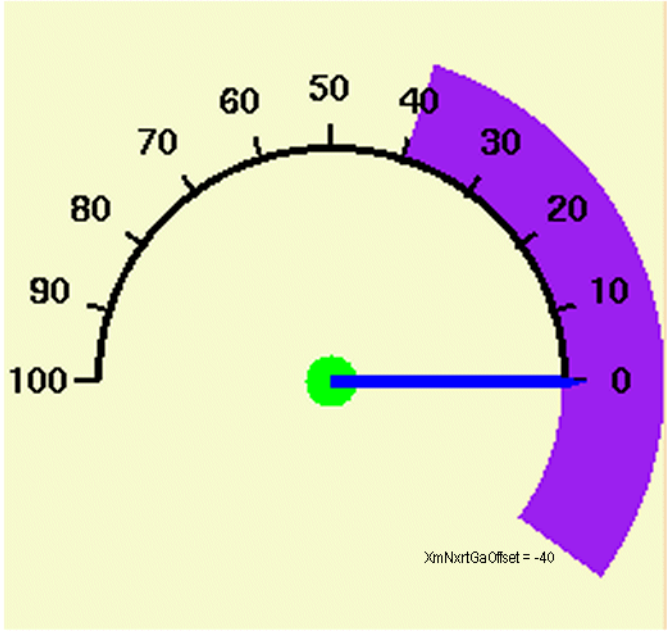


Figure 34 The same range object as in Figure 31 but with an added offset

Set the `XmNxrtGaOffset` resource to shift the range. As shown in Figure 34, a negative offset causes the range to begin before the start of the scale. The range can be offset a maximum of one scale range in either direction.

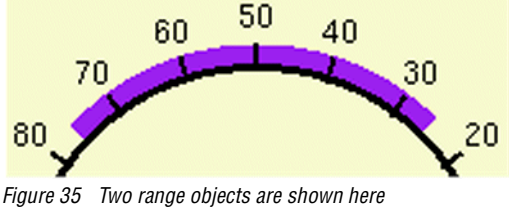


Figure 35 Two range objects are shown here

17.3 Sample Code

Set the resource values for the Range object using the samples below as a guide. The first example sets resources when the parent is a linear scale.

```
Widget lineardial; /* The parent widget for the Range object */
Widget ld_scale; /* The parent scale for the Range object */

/* Create Range */
range =
```

```
    XtVaCreateWidget("range",
                     XmNbackground,                          "green"
                     xmXrtGaRangeObjectClass,            lineardial,
                     XmNxrtGaScale,                        ld_scale,
                     XmNxrtGaInnerExtent,                 DVAL(0.9),
                     XmNxrtGaOuterExtent,                 DVAL(1.0),
                     XtVaTypedArg,                     XmNxrtGaImage,
                     XmRString,    "$XRTHOME/src/icons/shading.xpm",
                     strlen("$XRTHOME/src/icons/shading.xpm"+1),
                     XmNxrtGaImageLayout,  XRTGA_IMAGE_LAYOUT_TILED,
                     XmNxrtGaIsShowing,                        True,
                     XmNxrtGaOffset,                      DVAL(0.0),
                     XrtGaShape,              XRTGA_SHAPE_RECTANGLE,
                     XmNxrtGaStartValue,                  DVAL(0.0),
                     XmNxrtGaStopValue,                  DVAL(10.0),
                     NULL);
```

Use these resources when the parent is a circular scale:

```
Widget circulargauge; /* The parent widget for the Range object */
Widget cd_scale; /* The parent scale for the Range object */

/* Create Range */
range =
    XtVaCreateWidget("range",
                     XmNbackground,                          "blue"
                     xmXrtGaRangeObjectClass,         circulargauge,
                     XmNxrtGaScale,                        cd_scale,
                     XmNxrtGaInnerExtent,                 DVAL(0.9),
                     XmNxrtGaOuterExtent,                 DVAL(1.0),
                     XmNxrtGaIsShowing,                        True,
                     XmNxrtGaOffset,                      DVAL(0.0),
                     XmNxrtGaStartValue,                  DVAL(0.0),
                     XmNxrtGaStopValue,                  DVAL(10.0),
                     NULL);
```

17.4 Resource Reference

This section lists all of the resources for the range object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the Range object. If this resource is not set then the foreground color of the parent widget is used.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The range's background image that is used if the type is image and the associated scale is linear. The image is drawn at the center of the range's defined area.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_TILED** **CSG**

Specifies the positioning of the background image. Valid values are `XRTGA_IMAGE_LAYOUT_CENTERED` and `XRTGA_IMAGE_LAYOUT_TILED`.

XmNxrtGaInnerExtent **double** **1.0** **CSG**

The inner extent of the range. If the associated scale is circular, this value is related to the `XmNxrtGaRadiusSize` value of scale. For example, for a circular scale, a value of 0.5 means the inner extent should begin one-half the distance between the scale's origin and the scale's radius. For linear horizontal scales, an inner extent of 0.2 means the range's inner extent is 0.2 times the scale's height away from the top of the scale (the scale's position determines the top left of the scale).

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide a given object.

XmNxrtGaOffset **double** **0.0** **CSG**

For circular ranges, the offset that the start value has from the range's start value. This value is used to shift range regions, possibly outside the scale's limits. This offset can be up to one scale range (that is, the difference between the associated circular scale's *max* value and *min* value) in either direction.

XmNxrtGaOuterExtent **double** **1.0** **CSG**

The outer extent of the range. If the associated scale is circular, this value is related to the `XmNxrtGaRadiusSize` value of scale. For example, a value of 1.5 means the outer extent is 1.5 times the scale's radius. For linear horizontal scales, an outer extent of 0.8 signifies that the range's outer extent is 0.8 times the scale's height away from the top of the scale (the scale's position determines the top left of the scale).

XmNxrtGaScale **XrtGaObject** **NULL** **CSG**

The circular or linear scale associated with the object. If this value is `NULL`, the corresponding range is not drawn.

XmNxrtGaShape **XrtGaShape** **XRTGA_SHAPE_RECTANGLE** **CSG**

Shape of the range. This resource is ignored if the associated scale is circular (in this case a circular band or annular is always drawn). Valid values are `XRTGA_SHAPE_RECTANGLE`, `XRTGA_SHAPE_CIRCLE`, `XRTGA_SHAPE_THERMOMETER`, `XRTGA_SHAPE_TANK`, and `XRTGA_SHAPE_IMAGE`.

XmNxrtGaStartValue **double** ***dynamic*** **CSG**

The start value of the range. This should be between the associated scale's *min* and *max* values. If not, the value actually displayed will be confined to the scale's range (that is, a value greater

than the maximum will be displayed at the *max* value and a value under the *min* value will be displayed at the location of the minimum). The default value is the scale's minimum value.

XmNxrtGaStopValue **double** ***dynamic*** **CSG**

The stop value of the range. This should be between the associated scale's min and max value. If not the value actually displayed will be confined to the scale's range (that is, a value greater than the maximum will be displayed at the *max* value and a value under the *min* value will be displayed at the position of the minimum). The default value is the scale's maximum value.

17.5 Procedures and Methods Reference

This section lists the procedures and methods.

XmCreateXrtGaRange()

Standard Motif-style function to create an `XrtGaNeedle` object:

```
Widget
XmCreateXrtGaRange(
                    Widget parent,
                    char *name,
                    ArgList arglist,
                    Cardinal argcount
);
```

`parent` is the parent of the `XrtGaRange` object to be created; `name` is the name given the newly created `XrtGaRange` object; `arglist` is the list of `XrtArgVal` resource/value pairs to provide to `XrtGaRange` upon creation, which can be `NULL`; `argcount` is the number of elements in `arglist`.

The function returns `NULL` if the object could not be created.

XmIsXrtGaRange()

```
XmIsXrtGaRange(w)
```

Macro that returns `True` if `w` is an `XrtGaRange` object or a subclass of an `XrtGaRange` object.

18

State Object

XrtGaState ■ *Notes on the State Object's Attributes*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

18.1 XrtGaState

A State is a visible object associated with a Switch object in a Circular Dial widget. When a switch is in its discrete mode each possible setting is called a *state*.

The state object's parent must be an `XrtGaCircularDial` widget. The circular scale to which the state belongs must be set as well. For these discrete dials, a list of state objects defines the allowable states. Each new state object is automatically added to its parents state list as it is created. This list is stored in the widget's `XmNxrtGaStateList` resource. This list is read-only and the user should not manipulate its contents.

The state's label is specified using the `XmNxrtGaText` resource. The default alignment policy for the text in the labels is inherited from the switch. You may override the default alignment policy on a state-by-state basis using the `XmNxrtGaAlignment` resource. See `XmNxrtGaAlignment` in this chapter's resource section for more details on the alignment policy.

In the default case, the states distribute themselves evenly between the scale's start and stop angles. You can override the default setting and position each state yourself by setting the `XmNxrtGaAngle` resource.

18.2 Notes on the State Object's Attributes

The font used for drawing labels on states is specified by the `XmNfontList` resource.

Placing States around the scale

By default, states are placed uniformly between the start and stop angles of a circular scale. For a given state, its angle is automatically calculated if its

XmNxrtGaAngleUseDefault resource is True.You can specify a different angle for the XmNxrtGaAngle resource, which has the side effect of setting XmNxrtGaAngleUseDefault to False. Resetting XmNxrtGaAngleUseDefault back to True causes the widget to calculate the angle once again.

Aligning text lines in labels

States are labeled using the XmNxrtGaText resource. The string may contain formatting symbols such as \n, and they may be centered or justified using the XmNxrtGaAlignment resource. An example of the three alignment choices can be seen in Figure 36. *state1* is left justified (XRTGA_ALIGNMENT_BEGINNING), *state2* and *state3* are center justified (XRTGA_ALIGNMENT_CENTER), *state4* and *state5* are right justified (XRTGA_ALIGNMENT_END).

Managing the appearance of lines

Each state has some angular position on the scale. You can draw a line in this radial direction and out to its label. The beginning of the line is specified using XmNxrtGaLineInnerRadius. The radial part of the line ends at the value specified by XmNxrtGaLineOuterRadius. From there, an additional line segment may be drawn by setting values for XmNxrtGaOffsetX and XmNxrtGaOffsetY. The effect is most easily seen for *state5* in Figure 36. The angle for *state5* is 180°, so the radial part of the line is lies horizontally. The line's outer radius value is 1.2, meaning that it extends 20% more than the scale's radius. From there, a *y*-offset of 90 pixels is added and an *x*-offset of 40 pixels is subtracted. Since *x* values increase to the right and *y* values increases downwards, this combination causes the line segment to extend down and to the left. It ends mid-way along the right side of the box that encloses the label for *state5*.

Placing State objects

By default, inner radius, outer radius, and offsets are controlled by the state through its XmNxrtGaLineInnerRadius, XmNxrtGaLineOuterRadius, XmNxrtGaOffsetType and XmNxrtGaOffsetDistance resources.

If a state's XmNxrtGaInnerRadiusUseDefault or XmNxrtGaOuterRadiusUseDefault is True, it retrieves its inner or outer radius value from the Circular Dial's XmNxrtGaLineInnerRadius and XmNxrtGaLineOuterRadius resources. If a state's XmNxrtGaOffsetUseDefault is True, it determines its offset-*x* and offset-*y* values based on the Circular Dial's XmNxrtGaOffsetType resource. If the offset type is XRTGA_OFFSET_TYPE_CIRCULAR, the offset-*x* and offset-*y* values are set to zero. If it is XRTGA_OFFSET_TYPE_HORIZONTAL or XRTGA_OFFSET_TYPE_VERTICAL, the XmNxrtGaOffsetDistance is used to place the state the same distance from a horizontal or vertical line through the center of the circular scale. See Figure 6 on page 26 for an illustration. If the offset type is XRTGA_OFFSET_TYPE_AUTOMATIC, offset-*x* and offset-*y* values are chosen to achieve a pleasing look.

Note that if XmNxrtGaAlignmentUseDefault is True, an appropriate text alignment is chosen based on the offset type. If any of inner radius, outer radius, offset-*x* or offset-*y* values is set on a state, its corresponding UseDefault resource is set to False.

The type of line is controlled by `XmNxrtGaLinePattern`. Various choices are illustrated in Figure 36. You can control whether the line is displayed at all using the `XmNxrtGaLineDisplay` resource.

Color

Use `XmNforeground` to set the color of the text label (in its un-highlighted state) for the State object.

Controlling visibility

Use the `XmNxrtGaIsShowing` resource to specify whether the state is showing.

18.3 Sample Code

The code sample in this section sets values on most of the resources of a State object. Figure 36 shows the result of giving a switch five states, each with different line patterns, inner radii, alignments, and offsets. Starting from *switch1*, the alignment settings are *beginning*, *center*, *center*, *end*, *end*.

```
Widget circular_dial; /* The parent circular dial */
Widget cd_scale; /* The state is attached to this scale */
Position xpos1 = 10, ypos1 = 10;
   /* Create States */
   state1 = XtVaCreateWidget("state1",
                           xmXrtGaStateObjectClass,      circular_dial,
                           XmNxrtGaCircularScale,             cd_scale,
                           XmNxrtGaAlignment,      XRTGA_ALIGNMENT_END,
                           XmNxrtGaAlignmentUseDefault,          False,
                           XmNxrtGaAngle,                     DVAL(0.0),
                           XmNxrtGaAngleUseDefault,              False,
                           XmNxrtGaLineDisplay,                   True,
                           XmNxrtGaLinePattern,    XRTGA_LPAT_DASH_DOT,
                           XmNxrtGaLineThickness,                    2,
                           XmNxrtGaLineInnerRadiusUseDefault,    False,
                           XmNxrtGaLineInnerRadius,          DVAL(0.4),
                           XmNxrtGaLineOuterRadiusUseDefault,    False,
                           XmNxrtGaLineOuterRadius,          DVAL(1.2),
                           XmNxrtGaOffsetX,                      xpos1,
                           XmNxrtGaOffsetY,                      ypos1,
                           XmNxrtGaOffsetUseDefault,             False,
                           XmNxrtGaText,                     "state 1",
                           COLOR_RES(XmNforeground,            "blue"),
                           XmNxrtGaIsShowing,                    True,
                           NULL);
```

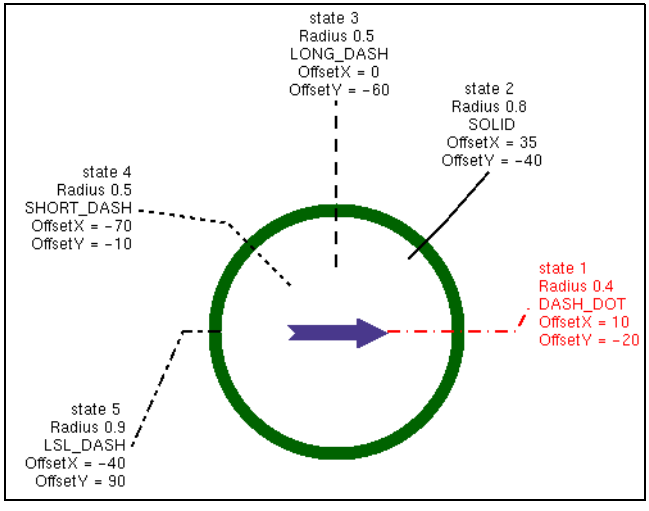


Figure 36 A switch with five states

18.4 Resource Reference

This section lists all of the resources for the State object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNfontList **XmFontList** ***"fixed"*** **CSG**

The font used for drawing any text for the state.

XmNforeground **Pixel** ***inherited*** **CSG**

The foreground color of the State object. If this resource is not set, the foreground color of the parent widget is used. This color is used for drawing text and lines in the object.

XmNxrtGaAlignment **XrtGaAlignment** **dynamic** **CSG**

This resource defines the alignment for the text associated with this state. If the corresponding `UseDefault` is `True`, the alignment is right for states on the left of the switch, left for states on the right of the switch, and centered for states above or below the switch. Setting this resource has the side effect of setting the corresponding UseDefault to False.

XmNxrtGaAlignmentUseDefault **Boolean** **True** **CSG**

If `True`, the `XmNxrtGaAlignment` resource uses its default value. This resource cannot be set to `False` if no value has yet been set for `XmNxrtGaAlignment` (That is, before a default has been calculated).

XmNxrtGaAngle **double** ***dynamic**** **CSG**

This resource defines the angle at which this state is drawn. By default, this angle is calculated from its associated circular scale's start angle, stop angle, number of states in the list, and where this state falls in that list. Setting this resource has the side effect of setting the corresponding UseDefault to False.

* `XmNxrtGaAngle` *gets its default value from its associated Circular Scale object.*

XmNxrtGaAngleUseDefault **Boolean** **True** **CSG**

If `True`, the `XmNxrtGaAngle` resource uses its default value. This resource cannot be set to `False` if no value has yet be set for `XmNxrtGaAngle` (that is, before a default has been calculated).

XmNxrtGaCircularScale **XrtGaObject** **NULL** **CSG**

The associated circular scale. If this value is `NULL`, the object will not be drawn.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the State object.

XmNxrtGaLineDisplay **Boolean** **True** **CSG**

If `True`, a line is displayed for this state.

XmNxrtGaLineInnerRadius **double** ***dynamic**** **CSG**

This resource defines the inner radius offset of this state. This resource's value is related to the radius of its parent's associated scale object. For example, a value of 0.5 means that the inner radius is half the scale's radius. When `XmNxrtGaLineInnerRadiusUseDefault` is `True`, the `XrtGaCircularDial`'s `XmNxrtGaLineInnerRadius` is used to determine this value. Setting this resource has the side effect of setting the `XmNxrtGaLineInnerRadiusUseDefault` to `False`.

* `XmNxrtGaLineInnerRadius` *gets its default value from its Circular Dial parent.*

XmNxrtGaLineInnerRadiusUseDefault **Boolean** **True** **CSG**

If `True`, the `XmNxrtGaLineInnerRadius` resource uses the default value obtained from its circular dial parent's `XmNxrtGaLineInnerRadius` resource. This resource cannot be set to `False` if no value has yet been set for `XmNxrtGaLineInnerRadius` (that is, before a default has been calculated).

XmNxrtGaLineOuterRadius **double** ***dynamic**** **CSG**

This resource defines the outer radius offset of this state. This resource's value is related to the radius of its parent's associated scale object. For example, a value of 1.5 means the outer radius is one and one-half times the scale's radius. When `XmNxrtGaLineOuterRadiusUseDefault` is `True`, the circular dial's `XmNxrtGaLineOuterRadius` resource is used to determine this value.

Setting this resource has the side effect of setting the XmNxrtGaLineOuterRadiusUseDefault to False.

* XmNxrtGaLineOuterRadius *gets its default value from its Circular Dial parent.*

XmNxrtGaLineOuterRadiusUseDefault **Boolean** **True** **CSG**

If True, the XmNxrtGaLineOuterRadius resource uses its default value obtained from its circular dial parent's XmNxrtGaLineOuterRadius resource. This resource cannot be set to False if no value has yet been set for XmNxrtGaLineOuterRadius, that is, before a value for the outer radius has been calculated.

XmNxrtGaLinePattern **XrtGaLinePattern** **XRTGA_LPAT_SOLID** **CSG**

Style of line for this state (if displayed). Possible values are:

XRTGA_LPAT_NONE	No line is displayed.
XRTGA_LPAT_SOLID	A solid line is displayed
XRTGA_LPAT_LONG_DASH	A line with long dashes is displayed.
XRTGA_LPAT_DOTTED	A dotted line is displayed.
XRTGA_LPAT_SHORT_DASH	A line with short dashes is displayed.
XRTGA_LPAT_LSL_DASH	A line with alternating long and short dashes is displayed.
XRTGA_LPAT_DASH_DOT	A line with alternating dashes and dots is displayed.

XmNxrtGaLineThickness **int** **1** **CSG**

The line thickness, if displayed. If the value of this resource is 0, the thinnest line possible is displayed.

XmNxrtGaOffsetUseDefault **Boolean** **True** **CSG**

This resource controls whether default values are used for the XmNxrtGaOffsetX and XmNxrtGaOffsetY resources. If either of these resources is changed by the user, XmNxrtGaOffsetUseDefault is set to False, and both offsets are no longer calculated by the widget. This resource cannot be set to False until values have been set for both the XmNxrtGaOffsetX and XmNxrtGaOffsetY resources.

XmNxrtGaOffsetX **Position** ***dynamic*** **CSG**

This offset is used to add the *x* direction offset for this state after the outer radius offset has been applied. By default, the Circular Dial's XmNxrtGaOffsetType and XmNxrtGaOffsetDistance resources are used to determine this value. Setting this resource has the side effect of setting XmNxrtGaOffsetUseDefault to False.

XmNxrtGaOffsetY **Position** ***dynamic*** **CSG**

This offset is used to add the *y* direction offset for this state after the outer radius offset has been applied. By default, the Circular Dial's `XmNxrtGaOffsetType` and `XmNxrtGaOffsetDistance` resources are used to determine this value. Setting this resource has the side effect of setting `XmNxrtGaOffsetUseDefault` to `False`.

XmNxrtGaText **String** **NULL** **CSG**

The string for this state.

18.5 Procedures and Methods Reference

XmCreateXrtGaState()

Standard Motif-style function to create an `XrtGaState` object:

```
Widget
    XmCreateXrtGaState (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaState` object to be created; *name* is the name to give the newly created `XrtGaState` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaState` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaState` object or `NULL` if the object could not be created.

XmIsXrtGaState()

```
XmIsXrtGaState(w)
```

Macro that returns `True` if `w` is an `XrtGaState` object or a subclass of an `XrtGaState` object.

19

Switch Object

XrtGaSwitch ■ *Notes on the Switch*
Sample Code ■ *Callbacks*
Resource Reference ■ *Procedures and Methods Reference*

19.1 XrtGaSwitch

A Switch object is used in a Linear Dial widget. Only one switch object per linear dial is allowed and it must be associated with the dial's circular scale.

You can choose a Switch object and specify one of four built-in types, or you can design your own switch and supply a drawing function for it.

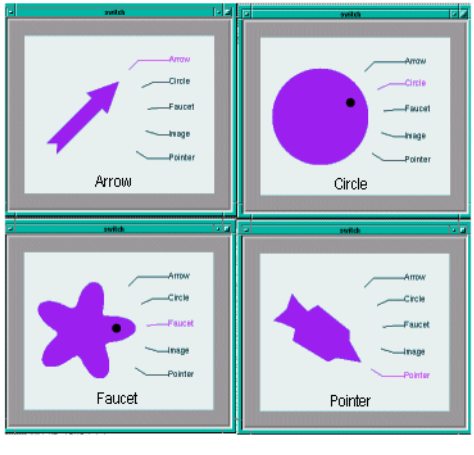


Figure 37 The four built-in shapes for a switch

19.2 Notes on the Switch

If the Circular Dial's XmNxrtGaSwitch resource is NULL, no switch or things that depend on the switch, such as markers and states, are drawn. No switch object is created for the parent widget by default. If a switch object is created and this resource is at its default value (NULL), this resource automatically points to the new object and this new object is used automatically by the widget. If the switch to which this resource is pointing is destroyed, the resource is set to NULL.

The value of a switch

A switch may be in one of two modes, continuous or discrete. In continuous mode, chosen by setting the XmNxrtGaContinuous resource to True, a switch may take on any value between the circular scale's *min* and *max* values.

For continuous switches it sometimes is still necessary to allow only incremental changes. The XmNxrtGaIncrementValue resource controls the allowable increments. For example, if the scale's *min* is zero and *max* is 100, an increment of 10 allows the switch to take on only values of 0, 10, 20,..., 100. If a value not equalling one of these incremental values is given to the object, it is rounded to the nearest increment. If the increment value is less than or equal to 0.0, no increment is used.

By setting the XmNxrtGaContinuous resource to False, only values 0, 1, 2 ... *n* -1, where *n* in the number of states, are allowed. Attempting to set some other value causes the switch to jump to the nearest allowed value.

Multi-turn switch

A switch can be put in a state in which multiple turns are possible. For this to happen, the switch must be in continuous mode by setting the XmNxrtGaContinuous resource to True, multiple turns must be enabled by setting the XmNxrtGaMultipleTurnEnabled resource to True, the multiple-turn resource, XmNxrtGaMultipleTurnValue, must have a value greater than one, and the associated circular scale's start angle must be equal to its stop angle so that use full circle is used. When enabled, the switch turns XmNxrtGaMultipleTurnValue revolutions from its associated scale's *min* value to its *max* value.

Using an image

It is possible to use an image for the switch. If you wish the image to rotate with the switch, set XmNxrtGaImageRotate to True.

The following example shows how you can set a background image as the switch:

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

void
set_image(Widget switch, char *file_name)
{
    XrtImage    image;
    XrmValue    from, to;

    from.addr = file_name;
    from.size = strlen(file_name) + 1;
    to.addr   = (caddr_t)&image;
```

```
        to.size   = sizeof(image);

        if (!XtConvertAndStore(image, XtRString,
                                          &from, XtRXrtImage, &to)) {
            fprintf(stderr, "Conversion of %s to XtRXrtImage failed\n",
                                                          file_name);
            image = NULL;
        }
        XtVaSetValues(switch,
                      XmNxrtGaImage,    image,
                      NULL);
        XrtPDSImageDestroy((XrtPDSImage)image);
    }
```

Note that images are copied by the widget, which is why you should destroy the image that was created by the converter. To remove an image, set the corresponding image resource to `NULL`.

See Common Elements in XRT/gauge for additional information on using images.

19.3 Sample Code

The sample code shown here attempts to set most of the resources associated with a Switch object.

```
    Widget circulardial;    /* The parent widget - a circular dial*/
    Widget cd_scale;        /* The parent circular scale*/
    XrtGaInteractionType interaction_type;

    interaction_type = XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE;

/* Create Switch */
    interaction_type = XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE;
    main_switch =
        XtVaCreateWidget("main_switch",
                         xmXrtGaSwitchObjectClass,         circulardial,
                         XmNxrtGaCircularScale,               cd_scale,
                         XmNxrtGaIsShowing,                       True,
                         XmNxrtGaRadius,                    DVAL(0.85),
                         XmNxrtGaContinuous,                      True,
                         XmNxrtGaIncrementValue,            DVAL(10.0),
                         XmNxrtGaMultipleTurnEnabled,             True,
                         XmNxrtGaMultipleTurnValue,          DVAL(2.0),
                         XmNxrtGaSwitchType,  XRTGA_SWITCH_TYPE_POINTER,
                         XmNxrtGaInteractionType,      interaction_type,
                         XmNxrtGaImageRotate,                     True,
                         XmNxrtGaValue,                     DVAL(0.0),
                         COLOR_RES(XmNbackground,      "darkslateblue"),
                         NULL);

    XtAddCallback(main_switch, XmNxrtGaValueChangedCallback,
                  cd_ValueCB, NULL);
```

Background images may be set along with other fallback resources:

```
static String fallbacks[] = {
    "*.cdial.background:                                    grey80",
    "*.xrtGaDebug:                                            True",
    "*.XmXrtGaSwitch.xrtGaImage:     $XRTHOME/src/icons/arrow3.xpm",
    NULL
};
```

Alternatively, you can set a background image through `XtVaSetValues` after defining a function that sets the image. An example of this approach is given in the previous section.

19.4 Callbacks

The `XmNxrtGaValueChangedCallback` callback can be registered from the Switch object. It is a list of callbacks called when the value changes. A pointer to an `XmNxetGaValueChangedCallbackStruct` is passed to each. See Chapter 22 for details.

19.5 Resource Reference

This section lists all of the resources for the Switch object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the Switch object. If this resource is not set then the foreground color of the parent widget is used.

XmNxrtGaCircularScale **XrtGaObject** **NULL** **CSG**

The circular scale associated with the Switch object. If this value is `NULL`, the Switch object will not be drawn.

XmNxrtGaContinuous **Boolean** **True** **CSG**

If `True`, the switch treats its range of values as a continuum and can take on any value between its minimum and its maximum, otherwise it is discrete and steps from one listed value to the next, but cannot take on an intermediate value, that is, one that is not in its list of values. The list of values in this case is 0, 1, 2, ... *n*-1, where *n* is the number of states.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDrawFunc **XrtGaDrawFunc** **NULL** **CSG**

Object drawing function. If this value is not `NULL`, it will be used to draw the object. `XmNxrtGaSwitchType` resources will be ignored.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

The switch's background image. If the switch's type is image and this resource is non-null, the image is drawn centered at the associated circular scale's origin. The value of the XmNxrtGaImageRotate resource determines whether the image will rotate as the switch's value changes. If you require the image to track the switch's value, use a marker instead.

XmNxrtGaImageRotate **Boolean** **True** **CSG**

Specifies whether the image is to be rotated as the angle changes.

XmNxrtGaIncrementValue **double** **0.0** **CSG**

For continuous switches it sometimes is still necessary to allow only incremental changes. For example, if the scale's *min* is zero and *max* is 100, an increment of 10 allows the switch to take only values of 0, 10, 20,..., 100. If a value not equalling one of these incremental values is given to the object, it is rounded to the nearest increment. If the increment value is less than or equal to 0.0, no increment is used.

XmNxrtGaInteractionType **XrtGaInteractionType** **XRTGA_INTERACTION_TYPE_NONE** **CSG**

The type of mouse interaction with the object. The possible values are `XRTGA_INTERACTION_TYPE_NONE`, `XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE`, and `XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE`.

`XRTGA_INTERACTION_TYPE_NONE`	Clicking and dragging have no effect.
`XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE`	Dragging causes the switch to follow the position of the mouse as long as it is within the parent widget.
`XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE`	Clicking causes the switch to follow the position of the mouse as long as it is within the parent widget.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the Switch object.

XmNxrtGaMultipleTurnEnabled **Boolean** **False** **CSG**

Allow the switch rotate multiple times in going from *min* to *max*. This resource is only used when `XmNxrtGaContinuous` is `True`, and other conditions are met, otherwise it is ignored. See the description under `XmNxrtGaMultipleTurnValue` for more details.

XmNxrtGaMultipleTurnValue **double** **1.0** **CSV**

To use this resource,

1. The switch must be in continuous mode (`XmNxrtGaContinuous` is `True`)
2. Enable multiple turns (`XmNxrtGaMultipleTurnEnabled` is `True`)
3. Set the associated circular scale's start angle to its stop angle, that is, use the full circle

4. Set this resource to a value greater than 1.0. For example, a value of 2.0 enables the switch to make two full rotations as it goes from its minimum value to its maximum value.

Note that any associated range or ticks on the circular dial will not be drawn to correspond to the multiple turn value.

XmNxrtGaRadius **double** **0.75** **CSG**

The outer extent of the switch object. This value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale. For example, a value of 0.5 means the radius of the switch object is half the size of the scale's radius.

XmNxrtGaSwitchType **XrtGaSwitchType** **XRTGA_SWITCH_TYPE_CIRCLE** **CSG**

The switch's type. Valid values are `XRTGA_SWITCH_TYPE_CIRCLE`, `XRTGA_SWITCH_TYPE_POINTER`, `XRTGA_SWITCH_TYPE_ARROW`, `XRTGA_SWITCH_TYPE_FAUCET`, or `XRTGA_SWITCH_TYPE_IMAGE`.

If you wish to use your own drawing function, see the section titled Providing Your Own Draw Function on page 181.

XmNxrtGaValue **double** ***dynamic*** **CSG**

The switch's current value. If the switch is in its continuous mode, this value should be between the associated scale's *min* and *max* values. If not, the value actually displayed will be confined to one endpoint or other of the scale's range. A value greater than *max* will be displayed at the *max* position, and a value less than *min* will be displayed at the *min* position.

If the switch is in its discrete mode, `XmNxrtGaValue` is an integer between 0 and *n*-1, where *n* is the number of states in the parent widget's state list. Any non-integer value given to the switch is rounded to an integer. Any out-of-range integer is ignored.

XmNxrtGaValueChangedCallback **XtCallbackList** **NULL** **CSG**

A list of callbacks called when the value changes. A pointer to an `XmNxetGaValueChangedCallbackStruct` is passed to each.

19.6 Procedures and Methods Reference

XmCreateXrtGaSwitch()

Standard Motif-style function to create an `XrtGaSwitch` object:

```
Widget
    XmCreateXrtGaSwitch (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaSwitch` object to be created; *name* is the name to give the newly created `XrtGaSwitch` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaSwitch` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaSwitch` object or `NULL` if the object could not be created.

XmIsXrtGaSwitch()

```
XmIsXrtGaSwitch(w)
```

Macro that returns `True` if `w` is an `XrtGaSwitch` object or a subclass of an `XrtGaSwitch` object.

20

Text Object

XrtGaText ■ *Notes on the Text Object's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

20.1 XrtGaText

Text objects are general-purpose text boxes used to annotate Circular Dial, Circular Gauge, Linear Dial, Linear Gauge, and Percent widgets. Any number of them may be placed anywhere within the boundaries of the widget.

20.2 Notes on the Text Object's Resources

You can choose a location within the widget by specifying the location of the top left corner of the text box using resources `XmNx` and `XmNy`. The dimensions of the text box are kept in resources `XmNwidth` and `XmNheight`, whose values are read-only.

Setting the `XmNbackground` resource distinguishes the text area from the rest of the widget. You can choose a border type using `XmNxrtGaBorder` and adjust its width using `XmNxrtGaBorderWidth`.

The text itself has the color specified using the `XmNforeground` resource. Its font is specified using the `XmNfontList` resource.

You may provide a background image for the text box using `XmNxrtGaImage`. See Setting Images with XtConvertAndStore on page 186 for details. If you want to center the image within the box, set `XmNxrtGaImageLayout` to `True`. If you want to tile the image within the box, set `XmNxrtGaImageLayout` to `False`. To ensure that the text box is large enough to contain your image, set `XmNxrtGaImageMinimumSize` to `True`.

You can control whether a text object is drawn using the `XmNxrtGaIsShowing` resource.

Choosing the string type

You have two choices for the type of string that is used in a text box, a "normal" C string, or an `XmString`. If you wish to use plain C strings, set the value of the `XmNxrtGaStringMode` resource to `XRTGA_STRING_MODE_STRING`, and if you wish to use an `XmString`, set the `XmNxrtGaStringMode` resource to `XRTGA_STRING_MODE_XM_STRING`. The string itself is set in `XmNxrtGaString` or `XmNxrtGaXmString`, depending on which mode you have chosen.

Displaying values in a string

The text object gives you a handy way of displaying the value of any XRT/gauge object that has an `XmNxrtGaValue` resource. First, ensure that the `XmNxrtGaStringMode` resource is set to `XRTGA_STRING_MODE_STRING`, then associate the object possessing a value with the text object using the `XmNxrtGaValueObject` resource. Place a format specifier, such as `"%g"`, as the value of the `XmNxrtGaString` resource. The text object will retrieve the value from the object you specified and display it in the text box, formatted in the way you directed.

The value object can be a counter, indicator, item, needle, or switch. The value object will automatically notify the object to redisplay when its value changes.

Aligning text

The lines of text within a Text object may be centered, or it may be right or left justified. Set `XmNxrtGaAlignment` to `XRTGA_ALIGNMENT_CENTER`, `XRTGA_ALIGNMENT_END`, or `XRTGA_ALIGNMENT_BEGINNING` to center, right justify, or left justify text within its enclosing box.

Adding Text

A text object is an independent rectangular region that can be placed anywhere within the widget by specifying the top left pixel coordinates of the text box.

If you are adding text to a circular gauge widget, you can do the following:

```
text = XtVaCreateWidget("text",
                        xmXrtGaTextObjectClass,             cgauge,
                        XmNx,                                  100,
                        XmNy,                                   50,
                        XmNxrtGaString,              "Hello World",
                        XmNxrtGaAlignment,   XRTGA_ALIGNMENT_CENTER,
                        COLOR_RES(XmNforeground,"blue"),
                        COLOR_RES(XmNbackground,"wheat"),
                        NULL);
   MtCenterText(cgauge,text);
```

`MtCenterText` is a convenience method that centers a Text object horizontally within the widget. It is located in *$XRTHOME/src/tools/mtgauge.c*.

20.3 Sample Code

This code snippet show an association between the text object and a needle object whose value is to be displayed. There is no argument for the %7.2 format specifier. Instead, the object knows that the needle's current value is to be used.

```
Widget needle; /* A needle whose value is displayed in the text */
/* Create a basic text area as a label */
text = XtVaCreateWidget("text",
                        xmXrtGaTextObjectClass,parent,
                        XmNy,                                    10,
                        XmNxrtGaValueObject,                  needle,
                        XmNxrtGaStringMode, XRTGA_STRING_MODE_STRING,
                        XmNxrtGaString,       "Needle value: %7.2f",
                        XmNxrtGaAlignment,   XRTGA_ALIGNMENT_CENTER,
                        COLOR_RES(XmNbackground,          "grey80"),
                        COLOR_RES(XmNforeground,           "black"),
                        NULL);
```

20.4 Resource Reference

This section lists all of the resources for the Text object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** **inherited** **CSG**

Background color of the object. If this resource is not set then the foreground color of the parent widget is used.

XmNfontList **XmFontList** ***"fixed"*** **CSG**

The font used for drawing any text for this object.

XmNforeground **Pixel** ***inherited*** **CSG**

Foreground color of the object. If this resource is not set, the foreground color of the parent widget is used. This color is used for drawing text and lines in the object.

XmNheight **Dimension** ***dynamic*** **G**

This resource is read only because it is determined by the vertical size of the text.

XmNwidth **Dimension** ***dynamic*** **G**

Contains the current width of the text object in pixels.

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

The *x*-coordinate of the top left corner of the object.

XmNxrtGaAlignment **XrtGaAlignment** **XRTGA_ALIGNMENT_CENTER** **CSG**

Specifies how multiple lines of text should be aligned. Possible values are `XRTGA_ALIGNMENT_BEGINNING`, `XRTGA_ALIGNMENT_CENTER`, and `XRTGA_ALIGNMENT_END`.

XmNxrtGaAllowWidgetPositioning **Boolean** **False** **CSG**

Allow the text area positioning to be overridden by the widget. If this resource is `False`, the user has control of the position. However, if this resource is `True` and the text object's

XmNxrtGaValueObject resource points to an object of type XrtGaItem, then the item object changes the text area's position so that it is positioned according to the item's XmNxrtGaRadius resource.

XmNxrtGaBorder **XrtGaBorder** **XRTGA_BORDER_NONE** **CSG**

The border type of the object. The width of all borders is given by XmNxrtGaBorderWidth. Possible values are

XRTGA_BORDER_NONE	No border at all.
XRTGA_BORDER_PLAIN	Border drawn in the foreground color.
XRTGA_BORDER_3D_IN	Area appears inset.
XRTGA_BORDER_3D_OUT	Area appears raised.
XRTGA_BORDER_ETCHED_IN	Double line, area appears inset.
XRTGA_BORDER_ETCHED_OUT	Double line, area appears raised.
XRTGA_BORDER_FRAME_IN	Shadow in at edge, area appears framed.
XRTGA_BORDER_FRAME_OUT	Shadow out at edge, area appears framed.
XRTGA_BORDER_BEVEL	Area appears framed.
XRTGA_BORDER_SHADOW	Shadow drawn under area.

XmNxrtGaBorderWidth **Dimension** **2** **CSG**

Specifies the width of an object's border in pixels.

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to XtAppWarningMessage from the *Xt error* database. When False, only resource conversion errors are printed.

XmNxrtGaImage **XrtGaImage** **NULL** **CSG**

Text objects's background image.

XmNxrtGaImageLayout **XrtGaImageLayout** **XRTGA_IMAGE_LAYOUT_CENTERED** **CSG**

Specifies the positioning of the background image. Valid values are XRTGA_IMAGE_LAYOUT_CENTERED and XRTGA_IMAGE_LAYOUT_TILED.

XmNxrtGaImageMinimumSize **Boolean** **False** **CSG**

Specifies that the minimum size of the text object will be that of its background image, if it exists.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide a given object.

XmNxrtGaString **char *** **NULL** **CSG**

The string that is displayed in the text object. The resource supports multiline text. Use this resource if the `XmNxrtGaStringMode` resource is `XRTGA_STRING_MODE_STRING`.

XmNxrtGaStringMode **XrtGaStringMode** **XRTGA_STRING_MODE_STRING** **CSG**

The type of string used in the text object. Possible values are `XRTGA_STRING_MODE_STRING` and `XRTGA_STRING_MODE_XM_STRING`. If the mode is `XRTGA_STRING_MODE_STRING`, the normal string is used and the `XmNxrtGaString` resource points to that string. If the mode is `XRTGA_STRING_MODE_XM_STRING`, an `XmString` is used, and the `XmNxrtGaXmString` resource points to that `XmString`.

XmNxrtGaValueObject **XrtGaObject** **NULL** **CSG**

Associated value object. When non-null, the text object can query this object for its value using the `XmNxrtGaValue` resource. The text object then uses this value to replace any formatted operands in the `XmNxrtGaString` resource, which then behave like the operands in *printf*.

XmNxrtGaXmString **XmString** **NULL** **CSG**

The string that is displayed in the text object. The resource supports multiline text. Use this resource if the `XmNxrtGaStringMode` resource is `XRTGA_STRING_MODE_XM_STRING`.

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

The *y*-coordinate of the top left corner of the object.

20.5 Procedures and Methods Reference

This section lists the Text object procedures and methods.

XmCreateXrtGaText()

Standard Motif-style function to create an `XrtGaText` object:

```
Widget
    XmCreateXrtGaText (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaText` object to be created; *name* is the name to give the newly created `XrtGaText` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaText` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaText` object or `NULL` if the object could not be created.

XmIsXrtGaText()

```
XmIsXrtGaText(w)
```

Macro that returns `True` if `w` is an `XrtGaText` object or a subclass of an `XrtGaText` object.

21

Tick Object

XrtGaTick ■ *Notes on the Tick Object's Resources*
Sample Code ■ *Resource Reference*
Procedures and Methods Reference

21.1 XrtGaTick

Tick objects are visible signs used to place regularly spaced marks on a scale. Tick marks have quite a few resources. This permits you to control the placement of the tick marks and their shape, the number of ticks in an interval, tick labels, and the number of decimal places that the labels display.

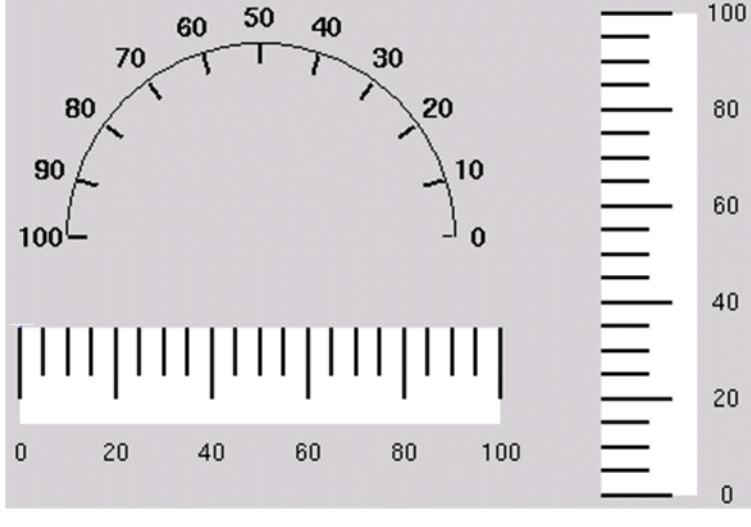


Figure 38 Some Tick objects and their associated labels

You specify where the tick marks are to begin and where they are to end. This allows you to place tick marks over a sub-portion of the scale. It is also possible to use an offset to place tick marks before the beginning or after the end of a scale.

21.2 Notes on the Tick Object's Resources

Associating a Tick object with a Scale

Tick objects need to know the scale for which they are to provide gradations. Associate a circular or linear scale with its Tick object using `XmNxrtGaScale`.

Setting the tick type

There are four built-in types: circle, diamond, rectangle, and triangle. Use `XmNxrtGaTickType` to select which.

Setting the Tick object's placement

Tick marks are normally placed at constant increments along a scale. The Tick object accomplishes this objective by using the associated scale's *min* and *max* values to determine appropriate values for the Tick object's *start*, *stop*, and *increment* values. If `XmNxrtGaPrecision` is set, this value is used as well, otherwise a default value for precision is determined. The effect is quite noticeable if `XmNxrtGaDrawLabels` is `True`. The number of labels changes as the scale is zoomed, becoming fewer during contraction so that the numbers on one label do not overlap adjacent labels, or becoming more numerous during expansion so that labeled marks do not become widely separated. All this happens when the Tick object automatically calculates ticks for you.

You can control the spacing if you wish by setting `XmNxrtGaAutomatic` to `False`. Then you control where the tick marks are to begin with `XmNxrtGaStartValue` and where they are to end with `XmNxrtGaStopValue`, then setting where to place the labels with `XmNxrtGaIncrementValue`.

Tick marks can extend outside the limits imposed by the start and stop values of the associated circular scale. The amount of offset is controlled by `XmNxrtGaOffset`.

Setting the Tick object's dimensions

The size of the tick marks is controlled with `XmNxrtGaInnerExtent` and `XmNxrtGaOuterExtent`. The values for both of these resources are numbers representing a proportion relative to the size of the associated circular or linear scale. For instance if the value set on `XmNxrtGaInnerExtent` is 1.0 and the related scale is circular, tick marks begin right at the circumference of the scale. They are drawn radially outward to the value set in `XmNxrtGaOuterExtent`, which is also given as a ratio based on the radius of the associated circular scale.

If the inner extent is less than the outer extent, these values are effectively switched. If the inner extent is equal to the outer extent, then no ticks are drawn. This is useful if you want labels but no tick marks. In non-automatic mode, that is, when `XmNxrtGaAutomatic` is `False`, if the stop value is less than the start value, no ticks or labels are drawn.

The width in pixels of the tick marks is specified in the XmNwidth resource.

Labeling tick increments

Labeling may be on or off, depending on the value of XmNxrtGaDrawLabels. In automatic mode, one of the ways the object manages the span between labeled tick marks is by using the XmNxrtGaPrecision resource's value. If you are controlling the placement of labeled tick marks, make sure you have set XmNxrtGaPrecision properly, as explained in the following section, "A note on precision". Note that setting the XmNxrtGaPrecision resource automatically sets the XmNxrtGaPrecisionUseDefault resource to False.

Place labels using XmNxrtGaLabelExtent. The interpretation of this resource's value depends on the type of scale, circular or linear, and, in the case of a linear scale, on its orientation. For a circular scale, an extent of 1.5 means that the label is placed outside the scale at a distance of one and one-half times the scale's radius. For a horizontal linear scale, distances are measured from the top, for vertical scales, distances are measured from the left.

Note that you can construct a custom type of labeling by writing your own function to supply the labels. The prototype for such a function is given in this chapter's resource reference under XmNxrtGaLabelFunc.

A note on precision

The value of the XmNxrtGaPrecision resource controls how tick labels are interpreted. There are three cases to consider:

- Positive values of XmNxrtGaPrecision indicate the number of places after the decimal place to include. For example, setting the XmNxrtGaPrecision resource to 3 causes the labels to be multiples of 0.001.
- Negative values indicate the positive number of zeros to use before the decimal place. For example, if the value of the XmNxrtGaPrecision resource is -3, numbering will be in multiples of 1000.
- A value of zero causes the labels to be integers.

Failure to set the precision resource may be the source of a misleading scale. As an example, setting the start value of a Tick object at -25, the stop value at 160, and allowing the default precision sets the precision to -1. In this case the value -25 is rounded to -20, resulting in a misleading scale. Setting the precision to 0 rectifies the problem because this specifies that digits in the units column must not be rounded.

21.3 Sample Code

The following example sets most of the Tick object's resource values. Many of these are the default values and are not strictly necessary. They are shown here for completeness only, but not as an example of good programming practice.

```
ticks = XtVaCreateWidget("ticks",
                         xmXrtGaTickObjectClass,cgauge,
                         XmNxrtGaScale,                    scale,
                         XmNxrtGaAutomatic,                False,
                         XmNxrtGaStartValue,            DVAL(0.0),
                         XmNxrtGaStopValue,           DVAL(100.0),
                         XmNxrtGaIncrementValue,       DVAL(10.0),
                         XmNxrtGaOffset,                DVAL(0.0),
                         XmNxrtGaPrecisionUseDefault,      False,
                         XmNxrtGaPrecision,                    0,
                         XmNwidth,                             2,
                         XmNxrtGaDrawLabels,                True,
                         XmNxrtGaIsShowing,                 True,
                         XmNxrtGaLabelExtent,           DVAL(1.15),
                         XmNxrtGaInnerExtent,            DVAL(0.9),
                         XmNxrtGaOuterExtent,            DVAL(1.0),
                         COLOR_RES(XmNbackground,         "black"),
                          NULL);
```

21.4 Resource Reference

This section lists all of the resources for the Tick object. 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 (*XtCreateWidget()*), **S** means it can be set (*XtSetValues()*), and **G** means it can be used in a get (*XtGetValues()*).

XmNbackground **Pixel** ***inherited*** **CSG**

Background color of the Tick object. If this resource is not set then the foreground color of the parent widget is used.

XmNfontList **XmFontList** **"fixed"** **CSG**

The font used for drawing any text for this object.

XmNwidth **Dimension** **1** **CSG**

The width in pixels of the widest part of the object. If this value is 0, draw a line of width 0 (the thinnest line possible) no matter what the type. Ticks whose type is circle ignore this resource.

XmNxrtGaAutomatic **Boolean** **True** **CSG**

Controls whether the tick object is functioning in automatic mode or in manual mode. If it is in automatic mode, the associated scale's *min* and *max* values and the value of `XmNxrtGaPrecision` are used to determine "pleasing" tick start, stop, and increment values. These values will automatically update the corresponding resources and any changes to these resources will be ignored, that is, the values reset to the automatic values. If the tick object is in

manual mode, the values in `XmNxrtGaStartValue`, `XmNxrtGaStopValue`, and `XmNxrtGaIncrement` value can be set by the user (they will default to the automatic values).

XmNxrtGaDebug **Boolean** **False** **CSG**

Specifies whether warning and error messages are to be printed to `XtAppWarningMessage` from the *Xt error* database. When `False`, only resource conversion errors are printed.

XmNxrtGaDrawLabels **Boolean** **False** **CSG**

If `True`, use the value of `XmNxrtGaPrecision` to draw a label for each tick value.

XmNxrtGaIncrementValue **double** ***dynamic*** **CSG**

The tick increment value. In automatic mode, this value is read-only. In manual mode, this value can be changed. If this value is less than or equal to 0, only the first tick is drawn.

XmNxrtGaInnerExtent **double** **1.0** **CSG**

The inner (towards the center of a circular scale or the top left of a linear scale) extent of each tick drawn. This value is related to the `XmNxrtGaRadiusSize` value of an associated circular scale or the height (if the orientation is horizontal) or width (if the orientation is vertical) of an associated linear scale. For circular scales a value of 0.9 means each tick's inner extent is 0.9 times the radius value of the scale away from the origin. For linear horizontal scales, an outer extent of 0.2 means each tick's outer extent is 0.2 times the scale's height away from the top of the scale. The scale's position determines the placement of the top left corner of the scale.

XmNxrtGaIsShowing **Boolean** **True** **CSG**

Specifies whether to display or hide the tick marks for this object.

XmNxrtGaLabelExtent **double** **1.0** **CSG**

The extent of labels (the center of each label). This value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale or the height (if the orientation is horizontal) or width (if the orientation is vertical) of an associated linear scale. For circular scales a value of 1.1 means each label's extent is 1.1 times the radius value of the scale away from the origin. For linear horizontal scales, a label extent of 0.5 means each label's extent is 0.5 times the scale's height away from the top of the scale.

XmNxrtGaLabelFunc **XtFunction** **NULL** **CSG**

A function the user can provide to produce a different label than the one the tick object would draw. It is called for each tick label. The prototype is:

```
static char * label_func(o, value, label, x, y)
    XrtGaObject o;
    double value;
    char *label;
    Position x, y;
```

`o` is the tick object; *value* is the value for the current tick label; `label` is the label that would have been drawn; `x` is the *x* position of the center of the label; `y` is the *y* position of the center of the label

XmNxrtGaOffset **double** **0.0** **CSG**

If the associated scale is circular, this resource determines the offset the start value has from the tick mark's start value. This value is used to shift tick areas possibly outside the scale's limits. This offset can be up to one scale range in either direction, that is, the associated circular scale's *max value - min value.* This resource has no effect if the associated scale is linear.

XmNxrtGaOuterExtent **double** **1.0** **CSG**

The outer extent of each tick drawn. Outer extent values increase towards the widget boundary for a circular scale or from top to bottom for a horizontal linear scale. This value is related to the `XmNxrtGaRadiusSize` value of the associated circular scale or to the height if the orientation is horizontal, or, if the orientation is vertical, the width of the associated linear scale. For circular scales a value of 1.1 means each tick's outer extent is 1.1 times the radius value of the scale in a direction away from the origin. For a linear horizontal scale, an outer extent of 0.8 means each tick's outer extent begins at 0.8 times the scale's height below the top of the scale.

XmNxrtGaPrecision **int** ***dynamic*** **CSG**

This resource defines the number of digits of precision after the decimal point to be used for labels. If its value is zero or less, the labels will all be integers.

A negative precision value indicates precision in powers of 10. For example, setting `XmNxrtGaPrecision` to -2 will cause all labels to be multiples of 100. Setting `XmNxrtGaPrecision` to 1 will will cause the labels to be multiples of 0.1. A side effect of setting this resource is to set the corresponding `XmNxrtGaPrecisionUseDefault` resource to `False`.

Failure to set the precision resource may be the source of a misleading scale. As an example, setting the start value of a Tick object at -25, the stop value at 160, and allowing the default precision sets the precision to -1. In this case the value -25 is rounded to -20, resulting in a misleading scale. Setting the precision to 0 rectifies the problem because this specifies that digits in the units column must not be rounded.

XmNxrtGaPrecisionUseDefault **Boolean** **True** **CSG**

When `True`, the widget or object calculates appropriate values for `XmNxrtGaPrecision` automatically at render time. When an explicit value is specified for `XmNxrtGaPrecision`, the widget or object sets this resource to `False`. You cannot set this resource to `False` unless a value for `XmNxrtGaPrecision` has been previously provided (either calculated or set).

XmNxrtGaScale **XmNxrtGaScale** **NULL** **CSG**

The associated circular or linear scale. If this value is `NULL`, the ticks that correspond to this Tick object will not be drawn.

XmNxrtGaStartValue **double** ***dynamic*** **CSG**

The value of the first tick (this should be between the associated scale's *min* and *max* values). In automatic mode, this value is read only. In manual mode, this value can be changed.

XmNxrtGaStopValue **double** ***dynamic*** **CSG**

An upper bound on tick values (this should be between the associated scale's *min* and *max* values). In automatic mode, this value is read only and is equal to the value of the last tick. In manual mode, this value can be changed.

XmNxrtGaTickType **XrtGaTickType** **XRTGA_TICK_TYPE_RECTANGLE** **CSG**

The shape of the tick mark, which can be rectangular, circular, diamond-shaped, or triangular. Possible values are `XRTGA_TICK_TYPE_RECTANGLE`, `XRTGA_TICK_TYPE_CIRCLE`, `XRTGA_TICK_TYPE_DIAMOND`, and `XRTGA_TICK_TYPE_TRIANGLE`.

21.5 Procedures and Methods Reference

This section lists the State object procedures and methods in alphabetical order.

XmCreateXrtGaTick()

Standard Motif-style function to create an `XrtGaTick` object:

```
Widget
    XmCreateXrtGaTick (
        Widget      parent,
        char *      name,
        ArgList     arglist,
        Cardinal    argcount
    )
```

parent is the parent of the `XrtGaTick` object to be created; *name* is the name to give the newly created `XrtGaTick` object; *arglist* is the list of `XrtArgVal` resource/value pairs passed to `XrtGaTick` on creation, which can be `NULL`; *argcount* is the number of elements in *arglist*.

The function returns the newly created `XrtGaTick` object or `NULL` if the object could not be created.

XmIsXrtGaTick()

```
XmIsXrtGaTick(w)
```

Macro that returns `True` if `w` is an `XrtGaTick` object or a subclass of an `XrtGaTick` object.

22

Programming User Interaction

User Interaction in XRT/gauge ■ *Overview of Translations and Actions*
Starting Widget/Object Interaction ■ *Updating Widget/Object Interaction*
Starting Translating and Zooming ■ *Updating Translating and Zooming*
The Counter Object's Reset Action ■ *Window Resizing*

This chapter describes the user-interaction features of XRT/gauge – how an end user can interact with an XRT/gauge widget and how an application can control that interaction.

22.1 User Interaction in XRT/gauge

By default, an end user can translate and zoom the linear and circular scales in Circular Gauge and Circular Dial widgets, and interact with Linear Gauge, Linear Dial, and Percent widgets. The LEDIcon widget allows its icons to be toggled via user interaction. Interaction can also be enabled in the Indicator, Needle, and Switch objects. The Counter object provides a reset button that permits resetting it to zero.

22.2 Overview of Translations and Actions

Each XRT/gauge widget, except XrtGaLEDNumerical, has a default translation table that maps user events to interactive action within the widget. You can customize the user's interaction capabilities through the following mechanisms:

- Translation tables – An application can override XRT/gauge's default translations by adding its own translations to the widget's translation table. This is useful for disabling particular interactions or changing the mouse/keyboard events that trigger an interaction. There is a list of translations and actions in the widget chapters. There are no translations defined for the LEDNumerical widget.

- Callbacks – An application can be notified each time a user interacts with the widget by registering callback procedures that XRT/gauge calls before, during, and after user action. This is useful for disallowing some interactions or constraining the allowed interaction. The following sections provide details on using each interaction's callback.

Three Interaction Stages

The XRT/gauge widget handles user interaction in three stages, each occurring in sequence:

1. Starting user interaction
2. Updating user interaction, and
3. Ending user interaction

Your application can be notified by callbacks during each stage so that particular interactions can be disallowed or constrained. The following sections describe each stage and its callbacks.

There are five types of interaction, Widget/Object interaction, which handles the interaction of needles, switches, indicators and icons, Translation, Zoom, ResetCounter, and the *Map* action. The first four have default translations registered for them with the various widgets.

22.3 Starting Widget/Object Interaction

The *ActionStart()* action is called to notify the application that an interaction on a widget or object is about to begin. Figure 39 illustrates this action routine and shows where any Interaction callbacks you add are called.

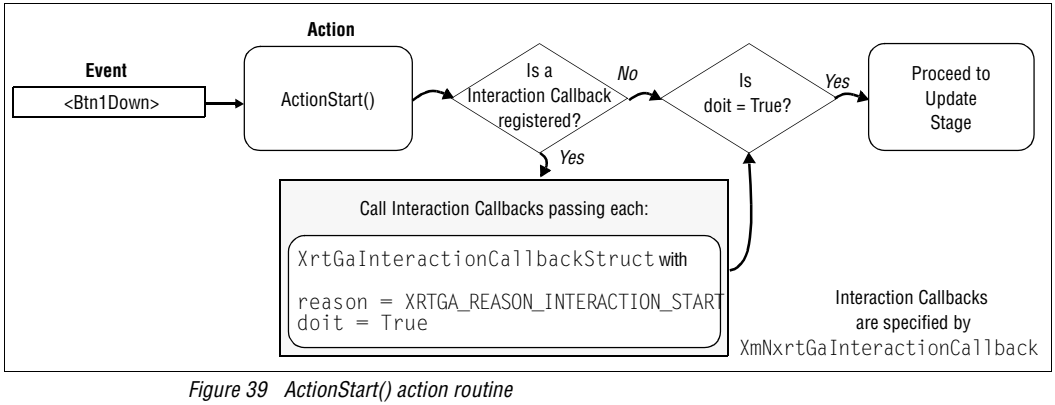


Figure 39 ActionStart() action routine

Disabling All User Interaction

You can use XmNxrtGaInteractionCallbackto disable any user interaction regardless of the translation table installed. Each callback procedure is passed a pointer to the following structure:

```
typedef struct {
    int      reason;    /* Read-only */
    XEvent   *event;    /* Read-only */
    Boolean  doit;      /* read/write when reason is not
                              XRTGA_REASON_INTERACTION_END. */
} XrtGaInteractionCallbackStruct;
```

Set the *doit* parameter of this structure to `False` (when the *reason* parameter is `XRTGA_REASON_INTERACTION_START`) to disable all user interactions. When *doit* is `False`, *ActionStart()* disables all "update" actions until the next *ActionStart()* action is called.

22.4 Updating Widget/Object Interaction

The *ActionMove()* action is called to notify the application that a user interaction is about to be updated. This action updates the XRT/gauge widget if the interaction has successfully passed through the *ActionStart()* action.

The *ActionMove()* action updates widget/object interaction of the XRT/gauge object or widget. The action updates the XmNxrtGaValue resource of a Needle object, Switch object, Indicator object, or an LEDIcon widget. Figure 40 illustrates this action routine and shows where any Interaction callbacks you add are called.

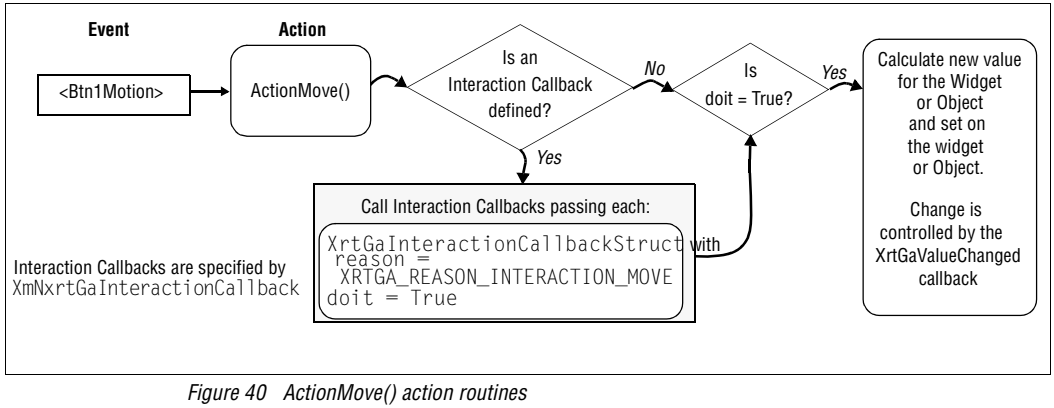


Figure 40 ActionMove() action routines

Controlling Interaction

You can use `XmNxrtGaInteractionCallback` to control translation or zooming. Each callback is passed a pointer to the following structure:

```
typedef struct {
    int      reason;    /* Read-only */
    XEvent   *event;    /* Read-only */
    Boolean  doit;      /* read/write when reason is not
                              XRTGA_REASON_INTERACTION_END. */
} XrtGaInteractionCallbackStruct;
```

Set the structure's doit parameter to `False` to disable the interaction.

22.4.1 Ending Widget/Object Interaction

The *ActionEnd()* interaction is called to notify the application that the action upon a LEDIcon widget or an Indicator, Needle, or Switch object has been completed.

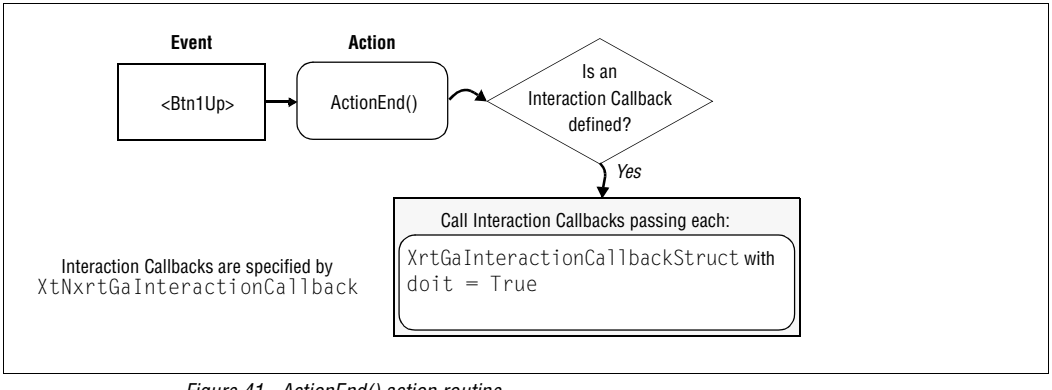


Figure 41 ActionEnd() action routine

22.5 Starting Translating and Zooming

The *TranslateStart()* and *ZoomStart()* actions are called to notify the application that a translate or zoom operation on a particular scale is about to begin. Figure 42 illustrates this action routine and shows where any Transform callbacks you add are called.

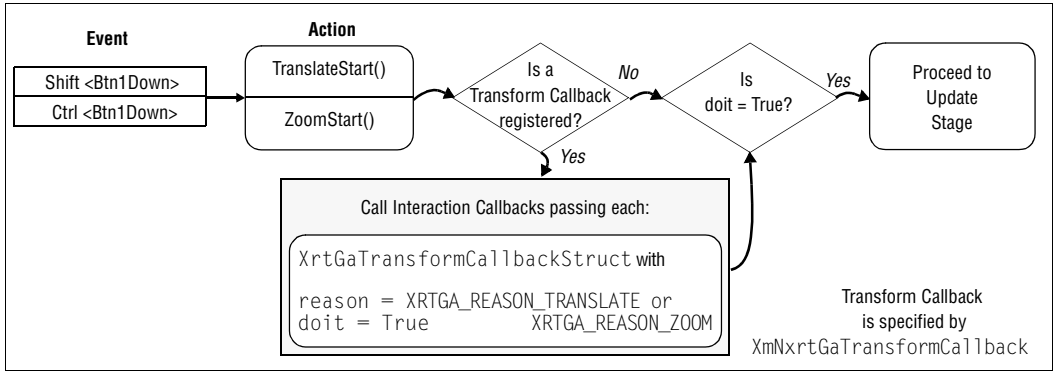


Figure 42 ActionStart() action routine

Monitoring or Disabling Translation and Zooming

You can use `XmNxrtGaTransformCallback` to monitor or constrain the translation and zooming process, or to disable it altogether. Each callback procedure is passed a pointer to the following structure:

```
typedef struct {
    int         reason          /* read-only */
    XEvent*     event           /* read-only */
    String*     params          /* read-only */
    Cardinal    nparams         /* read-only */
    Dimension   new_size        /* New radius size
                                   for circular scales. */
    Dimension   new_height      /* new height for linear scales */
    Dimension   new_width       /* new width for linear scales */
    Position    new_x           /* new x location*/
    Position    new_y           /* new y location*/
    Boolean     doit            /* True, Set to False
                                   to disallow the operation */
} XrtGaTransformCallbackStruct
```

The *reason* element indicates at what point the callback is being called; see Section XrtGaReason on page 224 for the possible values. The *event* element specifies the event. The *params* element is a list of parameters passed to the action. The *nparams* element is the number of parameters in the *params* list. The *new_size* element contains the new value for the radius of the circular scale. The *new_height* and *new_width* elements are the new width and height of for the linear scale. The *new_x* element contains the new position (origin for a circular scale and top left for a linear scale). Similarly, the *new_y* element contains the new value for the vertical component of the scale's position.. The *doit* element allows or disallows the operation. **Note:** if any parameter is -1, it means that the value is not to be changed, or is unused in the context of the current action.

`XmNxrtGaTransformCallback` is called when the end user attempts to move or zoom the scale. It is called from the *ZoomStart()*, *TranslateStart()*, *ZoomMove()*, and *TranslateEnd()* actions with the *reason* set to `XRTGA_REASON_ZOOM` and `XRTGA_REASON_TRANSLATE_MOVE`.

22.6 Updating Translating and Zooming

TranslateUpdate() and *ZoomUpdate()* are called to notify the application that a translate or zoom operation on a particular scale is about to be updated. None of these routines updates the XRT/gauge widget unless the interaction has successfully passed through the *TranslateStart()* or *ZoomStart()* actions.

22.6.1 Translation and Graphics Zooming

The *TranslateUpdate()* action updates interactive translation of circular or linear scales with various XRT/gauge widgets. Similarly, the *ZoomUpdate()* action updates an end user's interactive zooming of those scales. These routines alter the scale's resources. Figure 43 illustrates these action routines and shows where any Transform callbacks you add are called.

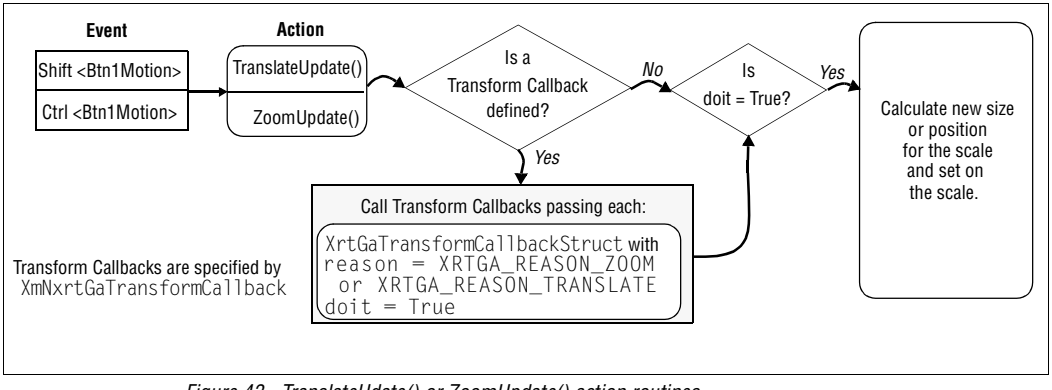


Figure 43 TranslateUdate() or ZoomUpdate() action routines

Controlling Interaction

You can use `XmNxrtGaTransformCallback` to control translation or zooming. This structure is passed to the `XmNxrtGaTransformCallback`. If any of the parameters have value "-1", it means that the value has not changed and will not change. Each callback is passed a pointer to the following structure:

```
typedef struct {
    int         reason;     /* read only */
    XEvent      *event;     /* read only */
    String      *params;    /* read only */
    Cardinal    nparams;    /* read only */
    Dimension   new_size;   /* New radius size
                                        for circular scales */
    Dimension   new_height; /* New height for linear scales */
    Dimension   new_width;  /* New width for linear scales */
    Position    new_x;      /* New X location for both
                                  linear and circular scales */
    Position    new_y;      /* New Y location for both
                                  linear and circular scales */
    Boolean     doit;       /* Default: True. User sets to
                                False to disallow the operation */
} XrtGaTransformCallbackStruct;
```

Set the structure's `doit` parameter to `False` to disable the interaction. The parameters are explained in the previous section.

22.6.2 Translation and Zooming

The *TranslateEnd()* and *ZoomEnd()* transforms are called to notify the application that an widget or an object has been updated.

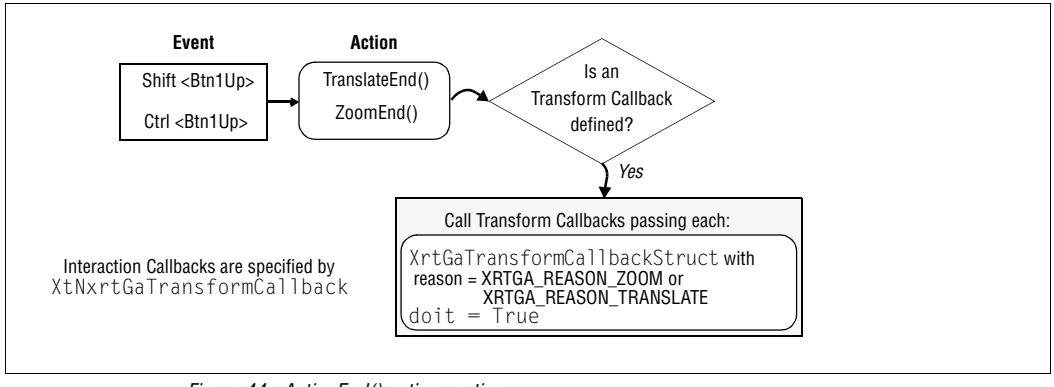


Figure 44 ActionEnd() action routine

22.7 The Counter Object's Reset Action

The *ResetCounter()* action is called to notify the application that an end user has performed a *Btn1Down* or a *Btn1Up* on a Circular Gauge or Circular Dial. Figure 45 illustrates this action routine and shows where any interaction callbacks you add are called. This action is called in order to reset a counter.

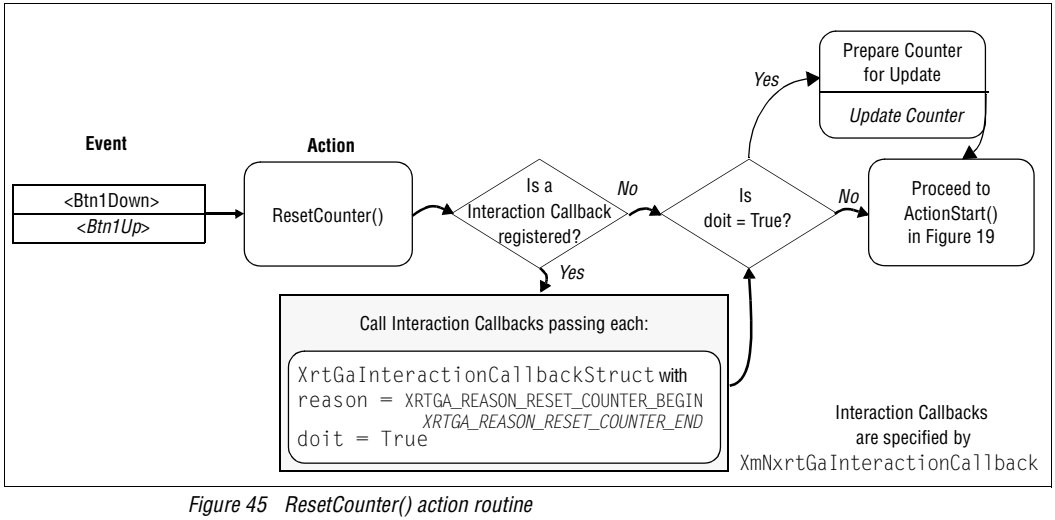


Figure 45 ResetCounter() action routine

The *ResetCounter()* action is called with either a `Begin` parameter or an `End` parameter. The behavior of *ResetCounter(End)* is quite similar to that of *ResetCounter(Begin)*. It is called on a *Btn1Up* instead of a *Btn1Down* event and both are shown in Figure 45. The items that pertain to *ResetCounter(End)* are shown using italicized text.

Note that *ResetCounter(End)* must be on the same reset button as *ResetCounter(Begin)*. If you do not want *Btn1* events to also influence a needle or a switch in the same widget as the reset button, disable the interaction on the needle or switch when the reason is `XRTGA_REASON_RESET_COUNTER_BEGIN` and enable it again when the reason is `XRTGA_REASON_RESET_COUNTER_END`. You can disable the reset button by setting `doit` to False when the reason is `XRTGA_REASON_RESET_COUNTER_END`.

22.7.1 Mapping

There are three things that work in conjunction to form the mapping mechanism in XRT/gauge. They are the *XrtGaMap()* function, the *Map()* action, and the `XmNxrtGaMapCallback`. The `XmNxrtGaMapCallback` callback enlists the aid of a `XrtGaMapCallbackStruct` to hold values that the *Map()* action provides to you. One of its fields is map, a pointer to `XrtGaMapResult`. The `XrtGaMapResult` structure is filled in when *XrtGaMap()* is called as a result of a *Map()* action. It holds values that tells the application which pixel the mouse was on when it was clicked, and to what values that pixel maps with respect to the scale in question.

The Map function

The *XrtGaMap()* function maps from pixel coordinates to circular or linear scale coordinates (that is, it maps from pixel space to a scale's coordinate space). It takes an scale object, (*x*, *y*) pixel coordinates, and a pointer to a map structure, calculates the widget's coordinates based on the scale, then writes the pixel coordinates and the scale's coordinates into the map structure.

```
Boolean
XrtGaMap(
    XrtGaObject     scale,
    Position        pix_x,
    Position        pix_y,
    XrtGaMapResult  *map
    )
```

XrtGaMap() returns `TRUE` if the scale is valid.

The `XrtGaMapResult` structure is defined as follows:

```
typedef struct {
    int         pix_x           /* The x-pixel value */
    int         pix_y           /* The y-pixel value */
    double      value           /* The scale value that corresponds
                                   to (pix_x, pix_y) */
    double      extent          /* The extent that corresponds to
                                   (pix_x, pix_y) */
    int         x_offset        /* The number of pixels that
                                  (pix_x, pix_y) is offset from the
                                   center or top left of the scale
                                   in the x direction */
```

```
    int         y_offset        /* The number of pixels that
                                   (pix_x, pix_y) is offset from the
                                   center or top left of the scale
                                   in the y direction */
    double      angle           /* The angle that corresponds to
                                   (pix_x, pix_y) - only used for
                                   circular scales*/
} XrtGaMapResult
```

The fields are broken down as follows:

`pix_x` The *x*-coordinate passed into *XrtGaMap()*.

`pix_y` The *y*-coordinate passed into *XrtGaMap()*.

`value` The value of `(pix_x, pix_y)` in the scale.

`extent` The extent along the radius, width, or height of the scale.

`angle` The angle for circular scales.

`x-offset` The distance along the *x*-axis from the scale origin.

`y-offset` The distance along the *y*-axis from the scale origin.

XrtGaUnmap()

The *XrtGaUnmap()* procedure is the opposite of *XrtGaMap()*. It maps from XRT/gauge widget coordinates to pixel coordinates. See XrtGaUnmap() on page 220 for more details.

The Map() action

A *Map()* action is provided so that you can capture window coordinates and convert them to scale values when the child object is a circular or linear scale. There is no default translation for this action, but one is easily defined. Note: you can use a command like

```
static char translations[] = "\
     <Btn1Up>:              Map()";
XtOverrideTranslations(widget,
                       XtParseTranslationTable(translations));
```

to add the *Map* action to the translation table. There is an example of using a Map callback in *$XRTHOME/src/gauge/examples/progress.c*.

The *Map()* action determines the widget coordinates that map to the pixel coordinates of a user event by calling *XrtGaMap()*. This produces an `XrtGaMapResult`, which is then passed to `XmNxrtGaMapCallback`. You can use the *Map()* action to find out which (x, y) values a user has clicked on. The way *Map()* works is illustrated in Figure 46.

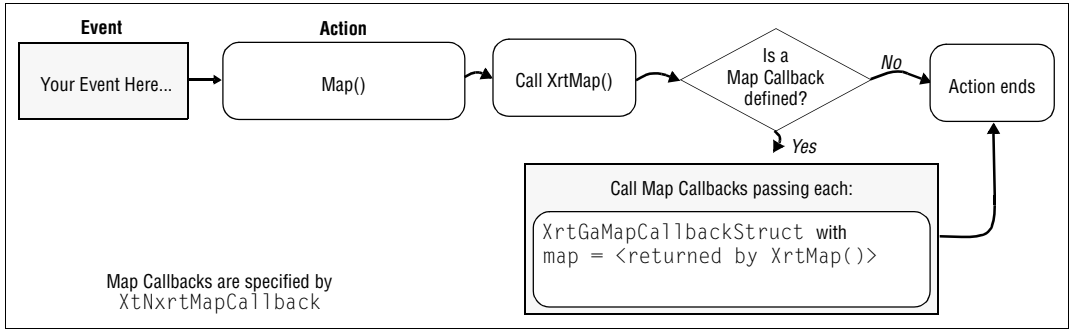


Figure 46 Map() Action Routine

Map callback

You must use `XtNxrtGaMapCallback` to register a callback to receive the results of the *Map()* procedure. Each callback is passed a pointer to the following structure:

```
typedef struct {
    int reason;         /* Read-only */
    XEvent *event;      /* Read-only */
    String *params;     /* Read-only */
    Cardinal nparams;   /* Read-only */
    XrtMapResult map;   /* Read-only */
} XrtGaMapCallbackStruct;
```

Changing any of the fields in this structure has no effect.

The map callback `XmNxrtGaMapCallback` resource in the Circular Gauge, Circular Dial, Linear Gauge, Linear Dial, and Percent widgets specifies a list of callbacks that are called to notify the application when an XRT/gauge widget has had a *Map()* action registered. A pointer to an `XrtGaMapCallbackStruct` is passed to each map callback.

An example follows in which a *Map* action is associated with three mouse operations and is invoked through the user-written *MapCB()* function. The callback function is used to synchronize the values held in objects of two related widgets, such as a Circular Gauge and a Linear Gauge. This way, both widgets are constrained always to display the same value. This code assumes that a function *GetObjectByName()* has been written that retrieves an item from a list by its name.

```
static char translations[] = "\
    <Btn1Up>:       Map()
    <Btn1Motion>:   Map()
    <Btn1Down>:     Map()";

Widget widget; /* A widget in the XRT/gauge family */
XtOverrideTranslations(widget,
                       XtParseTranslationTable(translations));
XtAddCallback(widgetObject,
              XmNxrtGaMapCallback,
              (XtCallbackProc)MapCB,
              (XtPointer) anotherWidgetsObject);
```

```
MapCB(w, client_data, cbs)
        Widget                  w;
        XtPointer               client_data;
        XrtGaMapCallbackStruct *cbs;
    {
        char                    string[20];
        Widget                  anotherWidgetsObject;
        XrtGaObject             object;
        XrtGaList               list;

        anotherWidgetsObject = (Widget) client_data;
        sprintf(string, "%.0f%%", cbs->map.value);

        /* Set Values on both progress charts */
        XtVaGetValues(w,
                      XmNxrtGaItemList,                        &list,
                      NULL);
        object = GetObjectByName(list, ITEM_NAME);
        if (object != NULL) {
            XtVaSetValues(object,
                          XmNxrtGaValue,       DVAL(cbs->map.value),
                          NULL);

        }

        XtVaGetValues(w,
                      XmNxrtGaTextList,                        &list,
                      NULL);
        object = GetObjectByName(list, TEXT_NAME);
        if (object != NULL) {
            XtVaSetValues(object,
                          XmNxrtGaString,                    string,
                          NULL);
        }

        XtVaGetValues(anotherWidgetsObject,
                      XmNxrtGaItemList,                        &list,
                      NULL);
        object = GetObjectByName(list, ITEM_NAME);
        if (object != NULL) {
            XtVaSetValues(object,
                          XmNxrtGaValue,       DVAL(cbs->map.value),
                          NULL);

    }

        XtVaGetValues(anotherWidgetsObject,
                      XmNxrtGaTextList,                        &list,
                      NULL);
        object = GetObjectByName(list, TEXT_NAME);
        if (object != NULL) {
            XtVaSetValues(object,
                          XmNxrtGaString,                    string,
                          NULL);
        }
    }
```

22.7.2 The Value Changed Callback

The value changed callback is defined as follows:

```
typedef struct {
    int         reason;                          /* read-only */
    XEvent      *event;                          /* read-only */
    double      value
    Boolean     doit;                       /* initially True */
} XrtGaValueChangedCallbackStruct;
```

It is called when the value changes in Item, Indicator, Needle, and Switch objects. The proposed new value is in `value`. This value may be changed so long as *reason* is `XRTGA_REASON_VALUE_CHANGE_BEGIN`, or may be disallowed by setting `doit` to `False`.

Note that when the callback is called with `reason` `XRTGA_REASON_VALUE_CHANGED_END`, the widget is just giving notification that the change has been completed.

22.7.3 Event Handling Procedures

The Xt Intrinsics provide a mechanism for registering interest in events such as mouse movement, mouse button clicks, keyboard clicks, etc. An XRT/gauge program can make use of the event handling mechanisms to react to events that happen over an XRT/gauge widget.

The following code will register the procedure *my_event_proc()* with the Xt Intrinsics. Whenever a mouse button event occurs over the gauge window, *my_event_proc()* will be called.

```
Widget      my_gauge;
XtAddEventHandler( my_gauge,
    ButtonPressMask | ButtonReleaseMask,
    FALSE, my_event_proc, NULL);
```

Event procedures are of the form:

```
void
my_event_proc(
    Widget      w,
    caddr_t     client_data,
    XEvent      *event
    )
{
    /* Deal with the event */
}
```

For more information on requesting and handling X events, consult your Xt Intrinsics and/or Motif programming manuals.

22.8 Window Resizing

Most applications should allow the user to resize a window containing an XRT/gauge widget, and have the widget adjust to the new window size.

The XRT/gauge widget can be located inside any of the Motif Composite widgets. Usually the Composite widget will determine the appropriate size for the XRT/gauge widget, and the widget will adjust automatically.

Expose & Resize Callbacks

An application can find out when an XRT/gauge widget is exposed or resized by specifying a callback function using the `XmNxrtGaExposeCallback` or `XmNxrtGaResizeCallback` resources. The callback is called with the widget handle and a pointer to any client data. The third callback argument, `call_data`, points to an `XrtGaCallbackStruct` which contains additional information about the callback. See Structures on page 226 for more details. The resize callback is called after the XRT/gauge widget is resized. The expose callback is called after the XRT/gauge widget is redrawn.

An application can use these callbacks to draw on top of the XRT/gauge widget image using Xlib functions. Another use is to adjust the widget resources depending on the size of the XRT/gauge widget.

The following code registers a resize callback function:

```
XtAddCallback(mygauge, XmNxrtGaResizeCallback,
    resize_proc, &my_data);
```

The following resize callback centers a Circular Gauge widget's scale when the widget is resized:

```
/* Callback to center a Circular Gauge Widget's scale */
void
resizeCB(w, client_data, call_data)
    Widget      w;
    XtPointer   client_data;
    XtPointer   call_data;
{
    Dimension       width, height;
    XrtGaObject     circular_scale;

    XtVaGetValues(w,
                  XmNwidth,                                  &width,
                  XmNheight,                                &height,
                  XmNxrtGaCircularScale,            &circular_scale,
                  NULL);

    if (!circular_scale) {
        return;
        }

    XtVaSetValues(circular_scale,
                  XmNxrtGaXOrigin,                        width/2,
                  XmNxrtGaYOrigin,                       height/2,
                  NULL);
}
```

23

Advanced Topics

Providing Your Own Draw Function ■ *Setting Images with XtConvertAndStore*

23.1 Providing Your Own Draw Function

You may supply your own draw function for Indicator, Marker, Needle, and Switch objects, and for the Linear Dial widget. User drawing functions have this prototype:

```
void
draw_func(object, user_data)
XrtGaObject object;
XtPointer   user_data;
```

This function gets called each time the widget wants to draw the given object. In the call, the user gets passed a pointer to an `XrtGaDrawData` structure from the widget through its `user_data` parameter:

```
typedef struct {
    Position    x;            /*X position of drawing origin */
    Position    y;            /*Y position of drawing origin */
    Dimension   width;        /*Width of draw object */
    Dimension   height;       /*Height of draw object */
    double      angle;       /*Angle at which object is to be drawn */
    Position    x_offset;     /*x distance from (x, y) */
    Position    y_offset;     /*y distance from (x, y) */
} XrtGaDrawData;
```

Based on the resources set on the object, the widget has calculated the position, dimensions, and angle at which the object is to be drawn. This information is passed in the `XrtGaDrawData` structure. The exact meaning of each parameter is dependent on the object (see section 23.1.1 on page 184).

The user can then use this information to draw the object in whatever way he/she wants. The user can also call *XrtGaGetUserDrawInfo()* to get widget-specific information about drawing (like the Window or the Pixmap into which the widget is currently drawing).

A mechanism has been provided for the user to draw a polygon that is scaled, rotated, and positioned based on the information in the `XrtGaDrawData` structure. To use this mechanism, the user must fill in an `XrtGaPolyDrawStruct` structure and pass it to the *XrtGaDrawPolyTransform()* method. The `XrtGaPolyDrawStruct` is defined as follows:

```
typedef struct {
    int             operations; /* Bitwise OR of XRTGA_* operations */
    Position        x;          /*X position to translate to. */
    Position        y;          /*Y position to translate to. */
    Dimension       width;      /*Width to scale to. */
    Dimension       height;     /*Height to scale to. */
    double          angle;      /*Angle to rotate to (0 degrees is E) */
    XPoint          *shape;     /*List of points */
    int             npoints;    /*Number of points in "shape". */
} XrtGaPolyDrawStruct;
```

In many situations, the user can just copy the information from the `XrtGaDrawData` structure directly to the `XrtGaPolyDrawStruct`. The user needs to define a polygon, whose origin is the center of rotation for the polygon. The polygon's span in the *x*-direction will be scaled to the width and its span in the *y*-direction will be scaled to the height specified in the `XrtGaPolyDrawStruct`. The angle specified starts at East and rotates clockwise. This is because the *y*-direction points to the bottom of the screen. The *x* and *y* fields specify the position within the widget to which the polygon is to be translated.

Note that the order of transformation is scale–rotate–translate. The user can control which transformations are done by specifying the operations flag. By doing a bitwise OR of the following transformation definitions you can specify any combination of

```
#define XRTGA_TRANSFORM_NONE  0x00
#define XRTGA_TRANSLATE       0x01
#define XRTGA_ROTATE          0x02
#define XRTGA_SCALE           0x04
#define XRTGA_TRANSFORM_ALL   (XRTGA_TRANSLATE | XRTGA_ROTATE |
                                                    XRTGA_SCALE)
```

Note that in the call to *XrtGaDrawPolyTransform()*, the widget will take care of all the other drawing details like drawing to the correct window or pixmap, or setting the background color of the polygon, thus there is no need to call *XrtGaUserDrawInfo()* if you use this mechanism.

Note that there are many examples of this in *$XRTHOME/src/gauge/util/draw_func.c*. These functions are the ones actually used by the widget to draw some of its predefined object types.

Example

By naming your function as the value of the `XmNxrtGaDrawFunc` resource, as shown in the call to `XtVaSetValues` below, the widget calls this function and supplies the proper parameters.

```
XtVaSetValues(needle,
          XmNxrtGaDrawFunc,DrawNeedle,
          NULL);

/* An example of a user-supplied drawing function */
```

```
DrawNeedle(object, user_data)
    XrtGaObject object; /* One of those mentioned above */
    XtPointer user_data;
{

    /*
    * Fancy arrow shape
    */
    static  Point arrow[] = {{10, 0}, {0, 10}, {110, 3}, {100, 10},
              {130, 0}, {100, -10}, {110, -3}, {0, -10}};

    XrtGaPolyDrawStruct poly;
    XrtGaDrawData       *dd;

    dd = (XrtGaDrawData *) user_data;

    poly.angle      = -dd->angle;
    poly.width      = dd->width;
    poly.height     = dd->height;
    poly.x          = dd->x;
    poly.y          = dd->y;
    poly.operations =
            (XRTGA_TRANSLATE | XRTGA_ROTATE | XRTGA_SCALE);
    poly.shape      = arrow;
    poly.npoints    = XtNumber(arrow);
    XrtGaDrawPolyTransform(object, (XtPointer)&poly);
}
```

23.1.1 The XrtGaDrawData Structure Parameters by Object

XrtGaIndicator

`x`	X position of the top left corner of where the indicator is to be drawn (taking into account the angle).
`y`	Y position of the top left corner of where the indicator is to be drawn (taking into account the angle).
`width`	The length of the indicator (*x*-direction -- maps to width in `XrtGaDrawPolyStruct`)
`height`	The width of the indicator (*y*-direction -- maps to height in `XrtGaDrawPolyStruct`)
`angle`	The angle the scale wants to draw the indicator. Note that this is the negative of the angle you want to provide *XrtGaDrawPolyTransform()* because the *y*-direction is downwards.
`x_offset`	Zero.
`y_offset`	Zero.

Note that the angle indicates the direction of the indicator:

0 degrees	Right
90 degrees	Bottom
180 degrees	Left
270 degrees	Top

XrtGaMarker

`x`	X origin of its associated circular scale
`y`	Y origin of its associated circular scale
`width`	The length of the marker (*x*-direction -- maps to width of `XrtGaDrawPolyStruct`)
`height`	The width of the marker (*y*-direction -- maps to height of `XrtGaDrawPolyStruct`)
`angle`	The angle the scale wants to draw the marker. Note that this is the negative of the angle you want to provide *XrtGaDrawPolyTransform()* because the *y*-direction is downwards.
`x_offset`	*x* offset to the inner radius of the marker from the scale's origin
`y_offset`	*y* offset to the inner radius of the marker from the scale's origin

XrtGaNeedle

`x`	X origin of its associated circular scale
`y`	Y origin of its associated circular scale
`width`	The length of the needle (*x*-direction -- maps to width in `XrtGaDrawPolyStruct`)
`height`	The width of the needle (*y*-direction -- maps to height in `XrtGaDrawPolyStruct`)
`angle`	The angle the scale wants to draw the needle. Note that this is the negative of the angle you want to provide to *XrtGaDrawPolyTransform()* because the *y*-direction is downwards.
`x_offset`	Zero, since the needle starts at the origin.
`y_offset`	Zero, since the needle starts at the origin.

XrtGaSwitch

x	X origin of its associated circular scale
y	Y origin of its associated circular scale
width	The scale's radius. (*x*-direction -- maps to width in `XrtGaDrawPolyStruct`)

height	The scale's radius (*y*-direction -- maps to height in `XrtGaDrawPolyStruct`)
angle	The angle the scale wants to draw the marker. Note that this is the negative of the angle you want to provide *XrtGaDrawPolyTransform()* because the *y*-direction is downwards.
x_offset	Zero. Since the switch starts at the origin.
y_offset	Zero. Since the switch starts at the origin.

Note that if the origin of the switch is within the polygon that you will have to scale polygon appropriately if you want its size to exceed the scale's radius. See the *XrtGaDrawSwitchPointer()* function in *draw_func.c* for an example of this.

23.2 Setting Images with XtConvertAndStore

An alternative to setting an image in a fallback resource is to use the *XtConvertAndStore()* function. An example follows.

1. First, construct a function that takes as parameters the widget with which the image is to be associated and the image's full path name:

```
GetColorImage(w, name)
             Widget     w; /* The parent widget */
             char       *name; /* The full path and filename
                               for the image, which may include
                               environment variables. */
{
    char        file_name[MAXPATHLEN + 1];
    XrtGaImage  image;
    XrmValue    from, to;

    MtExpandFilename(name, file_name); /* Environment variables are
                                          expanded here. */
    from.addr = file_name;
    from.size = strlen(file_name) + 1;
    to.addr  = (caddr_t)&image;
    to.size  = sizeof(image);

    if (!XtConvertAndStore(w, XmRString, &from, XmRXrtGaImage,&to))
        {
        fprintf(stderr,
                "Conversion of %s to XmRXrtGaImage failed\n",
                 file_name);
        image = NULL;
        }
    return(image);
}
```

2. Call the function:

```
XrtGaImage image;
image = GetColorImage(cimage_ldial,
                      "$XRTHOME/src/icons/teddybear_gift.jpg");
```

3. Use `XtVaSetValues` to assign the color to a resource. Here, it is assigned to the *on* color of a Linear Dial widget:

```
XtVaSetValues(linear_dial_widget,
              XmNxrtGaOnImage,  image,
              NULL);
XrtPDSImageDestroy(image);
```

   The call to `XtVaSetValues` copies the image, so the one created in the call to *XtConvertAndStore()* needs to be destroyed.

Note: the conversion library for JPEG images is not automatically registered with XRT/gauge. Register and enable JPEG image conversion with

```
XrtGaEnableImageType((XtPointer)XrtPDSJpegImageUtils);
```

24

PDS Utility Library

XrtPDSImage ■ Registering Image Types
Using Dynamic or Shared Libraries ■ XrtPDSList Object
Procedures and Methods Reference

24.1 XrtPDSImage

An XrtPDSImage is an opaque generic object which contains an image. It also encapsulates some of the image's properties such as the source filename, width, and height. The image format used internally is a Pixmap.

There are public functions that return the filename, width, height, clip mask (if available), and the Pixmap itself. None of these attributes can be set.

This allows the other XRT widgets to only have to deal with an image object. They do not have to worry about where the image came from or what format it was in originally. It also means that new image types can be added without changing any of the widget libraries.

24.1.1 Creating an XrtPDSImage

To create an XrtPDSImage from a Pixmap, call the *XrtPDSImageCreate()* method:

```
image = XrtPDSImageCreate(pixmap, display, width, height);
```

Note that *width* and *height* should be the actual width and height of the pixmap as there are no attempts to stretch, crop, or fit the pixmap to the given dimensions.

However, an XrtPDSImage is normally not directly created by the user from a Pixmap. Normally, the image is stored in a file in one of many possible formats. In XRT/gauge, the standard way to create an XrtPDSImage is to use the converters registered with the widget. By default, these converters are able to convert XPM, XWD, and GIF files to an XrtPDSImage. PNG and JPEG image support is also built into the widget, but must be explicitly enabled by registering the image type with the widget. Support for other image types can also be added (see Section 24.2.2).

24.2 Registering Image Types

XRT widgets capable of using `XrtPDSImage` objects maintain a list of registered image types. An unregistered or new image type can be registered using the function:

```
Boolean
XrtGaEnableImageType(
    XtPointer   functions
)
```

functions is a pointer to an `XrtPDSImageFuncs` structure. This structure is defined as follows:

```
typedef struct {
    XrtPDSImageNameCompareProc    check_name;
    XrtPDSImageHeaderCompareProc  check_header;
    XrtPDSImageCreateProc         create_image;
    XrtPDSImageType               type;
    XrtPDSImageOutputProc         output;
} XrtPDSImageFuncs
```

For example, to register JPEG input and output with the XRT/gauge widget:

```
#include <Xm/XrtGauge.h>
#include <Xm/XrtPDSJpeg.h>
XrtGaEnableImageType((XtPointer)xtXrtGaCircularGaugeClass,
                  (XtPointer)XrtPDSJpegImageUtils);
```

where *XrtPDSJpegImageUtils* is a pointer to the predefined `XrtPDSImageFuncs` structure for JPEG. Once you do this, you will be required to link in *libxrtjpeg.a* to your application.

Once an image type is registered, it is added to the list of available types. When a request is made to load an image from a file, the widget goes through the list of registered types calling `XrtPDSImageFuncs.check_name` to see if the filename of the file is a supported type. If this fails, the list is traversed again calling `XrtPDSImageFuncs.check_header` to see if the image type of the file can be identified from the first 250 bytes.

If an `XrtPDSImageFuncs` structure can be found that supports the image type stored in the file, then `XrtPDSImageFuncs.create_image` is called to create an XrtPDSImage from that image file.

Note that the `type` in the `XrtPDSImageFuncs` structure is just an `int` representing the type of image supported by this structure.

24.2.1 Predefined Image Support

The available predefined values are:

```
#define XRTPDS_XPM_IMAGE    0
#define XRTPDS_XWD_IMAGE    1
#define XRTPDS_GIF_IMAGE    2
#define XRTPDS_JPEG_IMAGE   3
#define XRTPDS_PNG_IMAGE    4
```

By default, support for reading XPM, XWD and GIF files and writing XPM and XWD files is automatically enabled in XRT/gauge, XRT/graph, and XRT/3d. PNG and JPEG image input/output can be added by linking the appropriate library and registering the image type. The JPEG library is called *libxrtjpeg.a* and the PNG library is called *libxrtpng.a* (it also requires the *libz.a* compression library).

Image support comes in two parts, the generic image object and the `XrtPDSImageFuncs` structure. The image object holds the converted image and the `XrtPDSImageFuncs` structure contains the methods that deal with reading (and possibly writing) the image.

Each image format has an associated `XrtPDSImageFuncs` structure. For example, `XrtPDSXpmImageUtils` is the structure for XPM images, and `XrtPDSJpegImageUtils` is the structure for JPEG images. To create an XrtPDSImage from a JPEG image, call `XrtPDSJpegImageUtils.create_image` and an image object containing the converted image is returned.

24.2.2 Custom Image Support

Adding support for reading and writing other image formats is accomplished by providing an `XrtPDSImageFuncs` structure with the appropriate fields filled out. The following shows the structure and the fields available:

```
typedef struct {
    XrtPDSImageNameCompareProc  check_name;
    XrtPDSImageHeaderCompareProccheck_header;
    XrtPDSImageCreateProc       create_image;
    XrtPDSImageType             type;
    XrtPDSImageOutputProc       output;
} XrtPDSImageFuncs;
```

For example, to add support for reading and writing Targa (.TGA) files, you first need to make the following definition and write the following functions:

```
/* Use a number greater than the maximum predefined type */
#define TARGA_IMAGE_TYPE 100

Boolean
CheckIsTargaFilename(
    String  filename;
    )

Boolean
CheckTargaMagicNumber(
    char    *buffer
    )

XrtPDSImage
CreateImageFromTargaFile(
    Widget  w,
    XImage  *image,
    String  filename,
    Boolean color,
    String  msg
    )

Boolean
```

```
WriteImageToTargaFile(
    Widget  w,
    XImage  *image,
    String  filename,
    Boolean color,
    String  msg
    )
```

Next, you need to create the `XrtPDSImageFuncs` structure and fill it out:

```
XrtPDSImageFuncs TargaImageUtilStruct = {
    (XrtPDSImageNameCompareProc) CheckIsTargaFilename,
    (XrtPDSImageHeaderCompareProc) CheckTargaMagicNumber,
    (XrtPDSImageCreateProc) CreateImageFromTargaFile,
    (XrtPDSImageType) TARGA_IMAGE_TYPE,
    (XrtPDSImage) WriteImageToTargaFile
};
XrtPDSImageFuncs *TargaImageUtils = &TargaImageUtilStruct
```

Finally, register this new Image type:

```
XrtGaEnableImageType(xtXrtGaCircularGaugeWidgetClass,
                     TargaImageUtils)
```

Targa images for all the image resources (such as `XtNxrtGaImage`) can now be read and written. You can also write Targa images by calling *XrtGaSaveImageGeneric()*:

```
XrtGaSaveImageGeneric(graph, TARGA_IMAGE_TYPE, "output.tga", NULL)
```

24.2.3 Writing the Image Functions

XrtPDSImageNameCompareProc

```
Boolean
CheckIsTargaFilename(
    String      file_name
    )
```

This function is passed an image filename and must return `True` if the filename has a valid extension or name (.tga for Targa files) and `False` if it does not.

XrtPDSImageHeaderCompareProc

```
Boolean
CheckTargaMagicNumber(
    char        *buffer
    )
```

This function is passed a buffer containing the first 250 bytes of the image file and it must determine if this file is a valid image file. This is generally done by looking for a magic number at the beginning of the file. For example, GIF files start with the characters “GIF87a” or “GIF89a”. This function returns `True` if the file is a valid image file and `False` if it is not.

XrtPDSImageCreateProc

```
XrtPDSImage
CreateImageFromTargaFile(
    Widget      w,
    String      file_name
    )
```

This is a real workhorse function. It is passed a widget and the name of an image file. It must create an XrtPDSImage from this file and return it. Generally, this involves the following steps:

1. Load the image in its native format.
2. Convert it to a Pixmap (the Xpm library has some handy functions for doing this).
3. Create an XrtPDSImage from this pixmap with *XrtPDSImageCreate()*

XrtPDSImageOutputProc

```
Boolean
WriteImageToTargaFile(
    Widget      w,
    XImage      *image,
    String      filename,
    Boolean     color,
    String      msg
    )
```

Another workhorse function. It is passed a widget, an XImage that is to be converted and saved, the filename of the file to which the image is to be saved, a Boolean indicating a black-and-white or color image and a message buffer (which may be `NULL`) to return any error messages in. If the buffer is `NULL`, error messages should be printed to `stdout` or `stderr`.

24.3 Using Dynamic or Shared Libraries

The JPEG and PNG image libraries are shipped compiled as static libraries. If you want to create dynamic or shared libraries from them, you have to obtain the source code (freely available on the Internet) and recompile them. Each of the image libraries has a custom XRT module added to it, but is otherwise unmodified. We provide this module in a compiled format ready to be added to a shared library (this module is in *$XRTHOME/lib-shared*).

Please note that this module is only available on the platforms that XRT PDS shared libraries are available on. See the release notes for specific platforms supported.

24.4 XrtPDSList Object

The XrtPDSList 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 XrtPDSList object is zero-based: the first element has index 0, the second element has index 1, and so on.

24.4.1 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()
{
    XrtPDSList list;
    int i, *ptr;

    list = XrtPDSListCreate(sizeof(int));
    i = 34;
    XrtPDSListAppend(list, &i);
    i = 46;
    XrtPDSListAppend(list, &i);
    ptr = XrtPDSListGetItem(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.

24.4.2 Creating and Destroying a List

To create a list, call the *XrtPDSListCreate()* method:

```
list = XrtPDSListCreate(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 *XrtPDSListCreate()*, call *XrtPDSListDestroy()*:

```
XrtPDSListDestroy(list);
```

All memory used by the list is freed.

Duplicating a List

To make a copy of an existing list, use the *XrtPDSListDuplicate()* method:

```
newlist = XrtPDSListDuplicate(oldlist);
```

This makes a copy of the list and all its elements, and returns a pointer to the copy. The original list is unchanged.

24.4.3 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 *XrtPDSListAppend()*:

```
index = XrtPDSListAppend(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 *XrtPDSListCreate()* . 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 *XrtPDSListInsertBefore()*:

```
newindex = XrtPDSListInsertBefore(list, oldindex, newitem);
```

XrtPDSListInsertBefore() 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, *XrtPDSListInsertAfter()* places a new item immediately after an existing list item:

```
newindex = XrtPDSListInsertAfter(list, oldindex, newitem);
```

Its parameters and return value are identical to those of *XrtPDSListInsertBefore()*.

If your list is sorted, you can add an item in its proper sorted position by calling *XrtPDSListAdd()*:

```
index = XrtPDSListAdd(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 24.4.6 on page 200.)

Removing and Deleting List Items

The XrtPDSList 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 *XrtPDSListRemove()*:

```
retpos = XrtPDSListRemove(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 *XrtPDSListRemoveAll()*:

```
XrtPDSListRemoveAll(list);
```

Note that neither of the above methods actually destroys the item memory. To destroy an item as well as remove it, call *XrtPDSListDelete()*:

```
retpos = XrtPDSListDestroy(list, position);
```

This method, like *XrtPDSListRemove()*, 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 *XrtPDSListDestroyAll()*:

```
XrtPDSListDestroyAll();
```

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 *XrtPDSListSetDestroyProc()*:

```
XrtPDSListSetDestroyProc(list, my_destroy_proc);
```

The destroy procedure is called whenever *XrtPDSListDestroy()* or *XrtPDSListDestroyAll()* is called, and must be defined to be of the following type:

```
void my_destroy_proc (
    XrtPDSList          object,
    XtPointer           item
    );
```

object is the list; *item* is the item to be destroyed.

The XrtPDSList object also enables you to destroy an object in a list by calling *XrtPDSListDestroyObject()*:

```
XrtPDSList              list, *objptr;

XrtPDSListDestroyObject(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 *XrtPDSListDestroyXtPointer()*:

```
XrtPDSList              list;
XtPointer               *item;

XrtPDSListDestroyXtPointer(list, item);
```

This method is passed the list and a pointer to the item to be destroyed.

24.4.4 Searching and Retrieving List Items

The XrtPDSList object defines several methods that provide the following search and retrieval operations:

- Determining whether an XrtPDSList 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 *XrtPDSListIsList()* method determines whether an object is actually an XrtPDSList:

```
Boolean                 is_list;
XrtPDSList              object;

is_list = XrtPDSListIsList(object);
```

XrtPDSListIsList() returns `True` if the object is an XrtPDSList 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:

- *XrtPDSListFind()*, which finds a list item matching a specified item using the XrtPDSList comparison procedure
- *XrtPDSListLookupItem()*, 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 `XRTPDS_LIST_ITEM_NOT_FOUND` if the item is not contained in the list.

To specify the comparison procedure to be used by *XrtPDSListFind()*, call *XrtPDSListSetCompareProc()*:

```
XrtPDSListSetCompareProc(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 *XrtPDSListGetCompareProc()*:

```
XrtPDSListItemCompareProc myproc;

myproc = XrtPDSListGetCompareProc(list);
```

The returned procedure is of type `XrtPDSListItemCompareProc`, which is defined to be the following:

```
typedef int (*XrtPDSListItemCompareProc) (
    XtPointer item1, XtPointer item2
);
```

This comparison procedure is also used in sorting. See section 24.4.6 on page 200 for a description of list sorting.

Retrieving an Item By Index

To retrieve an item given its position, call *XrtPDSListGetItem()*:

```
XtPointer               item;
XrtPDSList              list;
int                     index;

item = XrtPDSListGetItem(list, index);
```

XrtPDSListGetItem() 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 *XrtPDSListSetItem()*:

```
XrtPDSListSetItem(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 *XrtPDSListGetItems()*:

```
XrtPDSList              list;
XtPointer               array;

array = XrtPDSListGetItems(list);
```

If the list is empty, `NULL` is returned.

Searching for an Item

To search forward in a list for a specified item, call *XrtPDSListSearchForward()*:

```
int                     pos;
XrtPDSList              list;
int                     start_position;
XtPointer               item;

pos = XrtPDSListSearchForward(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, `XRTPDS_LIST_ITEM_NOT_FOUND` is returned.

To search backward in a list for a specified item, call *XrtPDSListSearchBackward()*:

```
pos = XrtPDSListSearchBackward(list, start_position, item);
```

The parameters and return value are the same as those of *XrtPDSListSearchForward()*.

Retrieving List Information

To obtain the number of items in a list, call *XrtPDSListGetItemCount()*:

```
int                     numitems;
XrtPDSList              list;

numitems = XrtPDSListGetItemCount(list);
```

This method is passed the list as its argument.

To obtain the size of an item in the list, call *XrtPDSListGetItemSize()*:

```
int                     itemsize;
XrtPDSList              list;

itemsize = XrtPDSListGetItemSize(list);
```

24.4.5 List Comparison Routines

The XrtPDSList 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 *XrtPDSListFind()*; to use one of these routines as the comparison procedure, pass it to *XrtPDSListSetCompareProc()*.

To compare two double-precision floating-point numbers, call *XrtPDSCmpDoubles()*:

```
XtPointer           double1, double2;
int                 retval;

retval = XrtPDSCmpDoubles(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.

XrtPDSCmpFloats() compares two single-precision floating-point numbers:

```
XtPointer           float1, float2;
int                 retval;

retval = XrtPDSCmpFloats(float1, float2);
```

The return value is the same as that of *XrtPDSCmpDoubles()*.

To compare two integers, call *XrtPDSCmpInts()*:

```
XtPointer           int1, int2;
int                 retval;

retval = XrtPDSCmpInts(int1, int2);
```

The return value is the same as that of *XrtPDSCmpDoubles()*.

To compare two strings, call *XrtPDSCmpStrings()*:

```
XtPointer           string1, string2;
int                 retval;

retval = XrtPDSCmpStrings(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.

XrtPDSCmpStrings() is case-sensitive. To perform a case-insensitive comparison of two strings, call *XrtPDSCmpStringsIgnoreCase()*:

```
XtPointer           string1, string2;
int                 retval;

retval = XrtPDSCmpStringsIgnoreCase(string1, string2);
```

The return value is the same as that of *XrtPDSCmpStrings()*.

24.4.6 Sorting a List

To sort the items in a list, call *XrtPDSListSort()*:

```
XrtPDSListSort(list);
```

The comparison procedure defined by *XrtPDSListSetCompareProc()* 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 *XrtPDSListSetCompareProc()*, see section 24.4.4 on page 196.

To determine whether a list is sorted, call *XrtPDSListIsSorted()*:

```
int                 is_sorted;

is_sorted = XrtPDSListIsSorted(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 *XrtPDSListMoveItemsBefore()*:

```
Boolean             ok;
XrtPDSList          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 = XrtPDSListMoveItemsBefore(list, destination, source,
    num_items);
```

XrtPDSListMoveItemsBefore() 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

XrtPDSListMoveItemsBefore() returns `True` if the move was successful, or `False` if an error occurred.

24.4.7 User-Defined Data

The XrtPDSList object enables you to define user data to be associated with the list. To define user data, a pointer to the user data must be passed to the *XrtPDSListSetUserData()* method:

```
XrtPDSList          list;
XtPointer           user_data;

XrtPDSListSetUserData(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 *XrtPDSListGetUserData()*:

```
user_data = XrtPDSListGetUserData(list);
```

If no user data is defined for this list, *XrtPDSListGetUserData()* returns `NULL`.

24.5 Procedures and Methods Reference

24.5.1 XrtPDSImage Methods

XrtPDSImageCopy()

Copies the image object.

```
XrtPDSImage
XrtPDSImageCopy(
    XrtPDSImage     image
    )
```

image is an image to be copied. Note that the internal Pixmap(s) are duplicated.

XrtPDSImageCreate()

Creates an XrtPDSImage with the associated Pixmap, Display, width and height.

```
XrtPDSImage
XrtPDSImageCreate(
    Pixmap          pixmap,
    Display         *display,
    int             width,
    int             height
    )
```

pixmap is a pixmap version of an XrtPDSImage; *display* is the X Display for which the image is created; *width* and *height* are the width and height of the XrtPDSImage created.

XrtPDSImageDestroy()

Destroys the image object and any memory associated with it.

```
void
XrtPDSImageDestroy(
    XrtPDSImage     image
    )
```

image is an image to be destroyed. Note that all of the image's internal Pixmaps are freed.

XrtPDSImageGetFilename()

Returns the filename this image was read from. Do not free this memory.

```
String
XrtPDSImageGetFilename(
    XrtPDSImage     image
    )
```

image is an image whose filename is returned.

XrtPDSImageGetWidth()

Returns the width of the image.

```
int
XrtPDSImageGetWidth(
    XrtPDSImage         image
    )
```

image is an image whose width is returned.

XrtPDSImageGetHeight()

Returns the height of the image.

```
int
XrtPDSImageGetHeight(
    XrtPDSImage         image
    )
```

image is an image whose height is returned.

XrtPDSImageGetColorDepth()

Returns the number of bits used to represent each color component for this image (generally 8).

```
int
XrtPDSImageGetColorDepth(
    XrtPDSImage         image
    )
```

image is an image whose color depth is returned.

XrtPDSImageGetClipMask()

Returns the clip mask for this image if it is available or `XmUNSPECIFIED_PIXMAP` if it is not.

```
Pixmap
XrtPDSImageGetClipMask(
    XrtPDSImage         image
    )
```

image is an image whose clip mask is returned.

XrtPDSImageGetImage()

Returns the Pixmap version for this image if it is available or `XmUNSPECIFIED_PIXMAP` if it is not.

```
Pixmap
XrtPDSImageGetImage(
    XrtPDSImage         image
    )
```

image is an image whose Pixmap version is returned.

XrtPDSImageIsSet()

Returns `True` if the Pixmap inside the image object is valid.

```
Boolean
XrtPDSImageIsSet(
    XrtPDSImage         image
    )
```

image is an image that returns `True` if the Pixmap is valid.

XrtPDSImageIsClipMastSet()

Returns `True` if the Pixmap for the clip mask inside the image object is valid.

```
Boolean
XrtPDSImageIsClipMaskSet(
    XrtPDSImage     image
    )
```

image is an image that returns `True` if the Pixmap for the clip mask is valid.

XrtPDSImageIsImage()

Returns `True` if the image is an XrtPDSImage.

```
Boolean
XrtPDSImageIsImage(
    XrtPDSImage     image
    )
```

image is an image that returns `True` if the image is an XrtPDSImage.

24.5.2 XrtPDSList Object Methods

XrtPDSCmpDoubles()

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 *XrtPDSListSetCompareProc()*.

```
int
XrtPDSCmpDoubles(
    XtPointer              double1,
    XtPointer              double2
    )
```

double1 and *double2* are the two numbers to be compared.

XrtPDSCmpFloats()

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 *XrtPDSListSetCompareProc()*.

```
int
XrtPDSCmpFloats(
    XtPointer       float1,
    XtPointer       float2
    )
```

float1 and *float2* are the two numbers to be compared.

XrtPDSCmpInts()

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 *XrtPDSListSetCompareProc()*.

```
int
XrtPDSCmpInts(
    XtPointer       int1,
    XtPointer       int2
    )
```

int1 and *int2* are the two numbers to be compared.

XrtPDSCmpStrings()

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 *XrtPDSListSetCompareProc()*.

```
int
XrtPDSCmpStrings(
    XtPointer       string1,
    XtPointer       string2
    )
```

string1 and *string2* are the two strings to be compared.

XrtPDSCmpStringsIgnoreCase()

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 *XrtPDSListSetCompareProc()*.

```
int
XrtPDSCmpStringsIgnoreCase(
    XtPointer       string1,
    XtPointer       string2
    )
```

string1 and *string2* are the two strings to be compared.

XrtPDSListAdd()

Adds an item to the list in its proper (sorted) position. The sort order is defined by the procedure returned by *XrtPDSListGetCompareProc()*. Returns the position index of the new node, or `XRTPDS_LIST_INVALID_POSITION` on failure.

```
int
XrtPDSListAdd(
    XrtPDSList      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 *XrtPDSListGetItemSize()*.

XrtPDSListAppend()

Appends an item to the end of a list. Returns the position index of the new node, or `XRTPDS_LIST_INVALID_POSITION` on failure.

```
int
XrtPDSListAppend(
    XrtPDSList      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 *XrtPDSListGetItemSize()*.

XrtPDSListCreate()

Creates a zero-length list of items, and returns it.

```
XrtPDSList
XrtPDSListCreate(
    size_t          item_size
    )
```

item_size is the intended size of the items in the list.

XrtPDSListDelete()

Deletes the item at the specified position. Returns the position of the deleted item.

```
int
XrtPDSListDelete(
    XrtPDSList      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 *XrtPDSSetDestroyProc()*), it is called when the item is deleted. (To remove an item without calling the destroy procedure, use *XrtPDSListRemove()*.)

XrtPDSListDeleteAll()

Deletes all the items from the list.

```
void
XrtPDSListDeleteAll(
    XrtPDSList      list,
    )
```

list is a list object whose items are to be deleted.

If a destroy procedure has been registered (by calling *XrtPDSSetDestroyProc()*), it is called each time an item is deleted. (To remove the items without calling the destroy procedure, use *XrtPDSListRemoveAll()*.)

XrtPDSListDestroy()

Destroys the specified list and frees all its allocated memory.

```
void
XrtPDSListDestroy(
    XrtPDSList      list
    )
```

list is a list object to be destroyed.

XrtPDSListDestroyObject()

Destroys an object in the specified list.

```
void
XrtPDSListDestroyObject(
    XrtPDSList      list,
    XrtPDSList      *object
    )
```

list is a list object containing the object to be destroyed; *object* is the object to be destroyed.

XrtPDSListDestroyXtPointer()

Destroys a non-object item in the specified list.

```
void
XrtPDSListDestroyXtPointer(
    XrtPDSList      list,
    XtPointer       *item
    )
```

list is a list object containing the item to be destroyed; *item* is the item to be destroyed.

XrtPDSListDuplicate()

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 *XrtPDSListSetDestroyProc()*.

```
XrtPDSList
XrtPDSListDuplicate(
    XrtPDSList      list
    )
```

list is a list object to be copied.

XrtPDSListFind()

Finds a list item matching a specified item (using the procedure returned by *XrtPDSListGetCompareProc()* to check for equality). Returns the position index of the item if found, or `XRTPDS_LIST_ITEM_NOT_FOUND` if the item is not found.

```
int
XrtPDSListFind(
    XrtPDSList      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 *XrtPDSIsSorted()* returns a non-zero value), 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.

XrtPDSListGetCompareProc()

Retrieves the procedure that the object uses for searching and sorting.

```
XrtPDSListItemCompareProc
XrtPDSListGetCompareProc(
    XrtPDSList          list
    )
```

list is a list object whose compare procedure is being returned.

XrtPDSListGetItem()

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
XrtPDSListGetItem(
    XrtPDSList          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.

XrtPDSListGetItemCount()

Retrieves the number of items stored in a list.

```
int
XrtPDSListGetItemCount(
    XrtPDSList          list
    )
```

list is a list object whose items are to be counted.

XrtPDSListGetItems()

Returns a pointer to the array of items in the list object, or `NULL` if the list is empty.

```
XtPointer
XrtPDSListGetItems(
    XrtPDSList          list
    )
```

list is a list object.

XrtPDSListGetItemSize()

Returns the size of each of the individual items in a list.

```
int
XrtPDSListGetItemSize(
    XrtPDSList          list
    )
```

list is a list object whose item size is being returned.

XrtPDSListGetUserData()

Gets the user data associated with an XrtPDSList object and returns it.

```
XtPointer
XrtPDSListGetUserData(
    XrtPDSList      list
    )
```

list is a list object whose user data is to be retrieved.

XrtPDSListInsertAfter()

Inserts a new item into a list after a specified position. Returns the index of the new node, or XRTPDS_LIST_INVALID_POSITION on failure.

```
int
XrtPDSListInsertAfter(
    XrtPDSList      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 *XrtPDSListGetItemSize()*.

XrtPDSListInsertBefore()

Inserts a new item into a list before a specified position. Returns the index of the new node, or XRTPDS_LIST_INVALID_POSITION on failure.

```
int
XrtPDSListInsertBefore(
    XrtPDSList      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 *XrtPDSListGetItemSize()*.

XrtPDSListIsList()

Returns True if the XrtPDSList being passed is an XrtPDSList object.

```
Boolean
XrtPDSListIsList(
    XrtPDSList      object
    )
```

object is an object to be checked.

XrtPDSListIsSorted()

Indicates whether the list is currently in a sorted state. Returns a non-zero value if the list is sorted.

```
int
XrtPDSListIsSorted(
    XrtPDSList              list
    )
```

list is a list object to be checked.

XrtPDSListLookupItem()

Locates the item in the specified list. This routine performs a strict memory compare and does *not* use the procedure returned by *XrtPDSListGetCompareProc()*. Returns the position of the matched item, or `XRTPDS_LIST_ITEM_NOT_FOUND` on failure.

```
int
XrtPDSListLookupItem(
    XrtPDSList              list,
    XtPointer               item
    )
```

list is a list to be searched; *item* is a pointer to the item to be located.

XrtPDSListMoveItemsAfter()

Moves a specified number of items from a source location to after the destination location.

```
Boolean
XrtPDSListMoveItemsAfter(
    XrtPDSList              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.

XrtPDSListMoveItemsBefore()

Moves a specified number of items from a source location to before the destination location.

```
Boolean
XrtPDSListMoveItemsBefore(
    XrtPDSList              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.

XrtPDSListRemove()

Removes the item at the specified position from a list. Returns the position of the removed item.

```
int
XrtPDSListRemove(
    XrtPDSList      list,
    int             position
    )
```

list is a list object; *position* is the position of the item to be removed.

The destroy procedure (registered by calling *XrtPDSSetDestroyProc()*), is *not* called when the item is removed. (To delete an item and call the destroy procedure, use *XrtPDSListDelete()*.)

XrtPDSListRemoveAll()

Removes all the items from a list.

```
void
XrtPDSListRemoveAll(
    XrtPDSList      list
    )
```

list is a list object whose items are to be removed.

The destroy procedure (registered by calling *XrtPDSSetDestroyProc()*), is *not* called when the item is removed. (To delete each item, calling the destroy procedure each time, use *XrtPDSListDeleteAll()*.)

XrtPDSListSearchForward()

Searches forward in a list, from a specified location, for a match to a specified item. (The procedure returned by *XrtPDSListGetCompareProc()* is used to perform comparisons.) Returns the position of the matched item, or `XRTPDS_LIST_ITEM_NOT_FOUND` on failure.

```
int
XrtPDSListSearchForward(
    XrtPDSList              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.

XrtPDSListSearchReverse()

Searches backward in a list, from a specified location, for a match for a specified item. (The procedure returned by *XrtPDSListGetCompareProc()* is used to perform comparisons.) Returns the position of the matched item, or `XRTPDS_LIST_ITEM_NOT_FOUND` on failure.

```
int
XrtPDSListSearchReverse(
    XrtPDSList              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.

XrtPDSListSetCompareProc()

Specifies the procedure that a list object uses for sorting.

```
void
XrtPDSListSetCompareProc(
    XrtPDSList                      list,
    XrtPDSListItemCompareProc       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 (*XrtPDSListItemCompareProc) (
    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*

XrtPDSListSetDestroyProc()

Specifies the procedure that the list object uses to destroy its items (if they were allocated in some special fashion).

```
void
XrtPDSListSetDestroyProc(
    XrtPDSList                  list,
    XrtPDSListItemDestroyProc   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 (*XrtPDSListItemDestroyProc) (
    XrtPDSList              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 *XrtPDSListSetUserData()*).

XrtPDSListSetItem()

Sets an item in a list. Returns `False` if the index specified is invalid.

```
Boolean
XrtPDSListSetItem(
    XrtPDSList              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.

XrtPDSListSetUserData()

Sets the user data associated with an XrtPDSList object.

```
void
XrtPDSListSetUserData(
    XrtPDSList              list,
    XtPointer               user_data
    )
```

list is a list object; *user_data* is the user data.

XrtPDSListSort()

Sorts the items in a list object. If the list is already sorted, this method does nothing.

```
void
XrtPDSListSort(
    XrtPDSList              list
    )
```

list is a list to be sorted.

Part II

Appendices

A

Macros and Methods

A.1 Macros

The following macros are provided for convenience in setting colors and double values. They are defined in *$XRTHOME/include/Xm/XrtGauge.h*

Macro Name	Definition
COLOR_RES(res, color)	XtVaTypedArg, res, XmRString, \ color, strlen(color) + 1
DVAL(value)	XrtGaDoubleToArgVal(value)

A.2 Methods

XrtGaArgValToDouble()

Converts an `XtArgVal` to a double. This procedure is the opposite of *XrtGaDoubleToArgVal()*. value is the `XtArgVal` to be converted.

```
double
XrtGaArgValToDouble(value)
                    XtArgVal value;
```

XrtGaDoubleToArgVal()

Converts a double to an `XtArgVal`. This procedure is the opposite of *XrtGaArgValToDouble()*. It is used for setting resources of type double in routines like `XtSetValues`, `XtVaSetValues`, and so on. `dval` is the double to be converted.

```
XtArgVal
XrtGaDoubleToArgVal(dval)
                    double dval;
```

Note that you can use *DVAL()* as a macro that expands to *XrtGaDoubleToArgVal()*. To use it you need to include *$XRTHOME/src/tools/mtgauge.h*.

XrtGaDrawPolyTransform()

Translates, scales and rotates a list of points as a polygon, then draws them. `w` is the widget in which the transform is to occur. `data` expects a pointer to an `XrtGaPolyDrawStruct` structure that is cast to an `XtPointer`.

```
void
XrtGaDrawPolyTransform(w, data)
                        Widget    w;
                        XtPointer data;
```

For more information on defining your own drawing function, see the section titled Providing Your Own Draw Function on page 181.

XrtGaEnableImageType()

Registers a new set of functions to be used when converting and unconverting images. The function returns `True` if the new set of functions was successfully registered and `False` if it was not. `functions` is a pointer to an `XrtPDSImageFuncs` structure. See XrtPDSImage on page 189 for more information.

```
Boolean
XrtGaEnableImageType(functions)
                    XtPointer functions;
```

XrtGaGetImage()

Returns a pointer to the XImage of a XRT/gauge widget. This may be useful for Xlib-level programmers who need access to the widget's XImage.

```
XImage *
XrtGaGetImage(gauge)
              Widget gauge;
```

gauge is the handle of the XRT/gauge widget to be output. *XrtGaGetImage()* returns `NULL` on failure. The XImage will be complete, even if the widget is partially obscured on the screen.

XrtGaMap()

Maps `pix_x` and `pix_y` to a value and an extent (returned in the `XrtGaMapResult` structure) based on the parameters of the scale. Returns `True` if the function succeeded (that is, the scale is valid). `scale` is the circular or linear scale for the mapping. `pix_x` is the *x* pixel position for the mapping. `pix_y` is the *y* pixel position for the mapping. `map` is the structure in which the value and extent, as well as other information, is returned.

```
Boolean
XrtGaMap(scale, pix_x, pix_y, map)
         XrtGaObject    scale;
         Position       pix_x;
         Position       pix_y;
         XrtGaMapResult *map;
```

XrtGaOutputXwd()

Outputs an X Window Dump representation of the gauge image to a file.

```
int
XrtGaOutputXwd(gauge, *fp, *msg)
               Widget gauge;
               FILE *fp;
               char *msg;
```

gauge is the handle of the XRT/gauge widget to be output. *fp* should point to a file which is already opened and ready for writing. If *XrtGaOutputXwd()* encounters problems, it returns a message in *msg* (*msg* should be at least 100 bytes). If *msg* is NULL, any error message will be written to *stderr*. *XrtGaOutputXwd()* returns 0 on failure and 1 on success. The `xwd` file will be complete, even if the widget is partially obscured on screen.

XrtGaSaveImageAsJpeg()

Outputs a JPEG representation of the gauge to a file. The JPEG image type must first be registered with XRT/gauge using *XrtGaEnableImageType()*.

```
Boolean
XrtGaSaveImageAsJpeg(gauge, filename, msg)
                     Widget gauge;
                     String filename;
                     String msg;
```

gauge is the handle of the gauge to be output. *filename* is the name of the file to which the image is to be written. If *XrtGaSaveImageAsJpeg()* encounters problems, it returns a message in *msg* (msg should be at least 100 bytes). If *msg* is `NULL`, any error message will be written to *stderr*. *XrtGaSaveImageAsJpeg()* returns `True` on success.

XrtGaSaveImageAsPng()

Outputs a PNG representation of the gauge to a file. The PNG image type must first be registered with XRT/gauge using *XrtGaEnableImageType()*.

```
Boolean
    XrtGaSaveImageAsPng(gauge, filename, msg)
                        Widget gauge;
                        String filename;
                        String msg;
```

gauge is the handle of the gauge to be output. *filename* is the name of the file to which the image is to be written. If *XrtGaSaveImageAsPng()* encounters problems, it returns a message in *msg* (msg should be at least 100 bytes). If *msg* is `NULL`, any error message will be written to *stderr*. *XrtGaSaveImageAsPng()* returns `True` on success.

XrtGaSaveImageGeneric()

Outputs a representation of the gauge to a file based on the image type. The image type must be registered with XRT/gauge using *XrtGaEnableImageType()* unless it is a preregistered type (XPM or XWD).

```
Boolean
XrtGaSaveImageGeneric(gauge, image_type, msg)
                    Widget gauge;
                    int image_type;
                    String filename;
                    String msg;
```

gauge is the handle of the gauge to be output. *image_type* is the image type. *filename* is the name of the file to which the image is to be written. If *XrtGaSaveImageGeneric()* encounters problems, it returns a message in *msg* (msg should be at least 100 bytes). If *msg* is `NULL`, any error message is written to *stderr*. *XrtGaSaveImageGeneric()* returns `True` on success.

XrtGaSetXmStringConverter()

Sets the `XmRXmString` resource converter to create formatted `XmString` values. This procedure must be called before any XRT/gauge widget is created.

```
void
XrtGaSetXmStringConverter(void)
```

XrtGaUnmap()

Maps the value and extent to a position `(pix_x, pix_y)` based on the parameters of the scale. The position is returned in the `XrtGaMapResult` structure. Returns `True` if the function succeeded, that is, the scale is valid and the value is between the *min* and *max* of the scale. `scale` is the circular or linear scale for the mapping. `value` is the scale's value for the mapping `extent`s in the radial direction for circular scales. For linear scales, the `extent`'s direction is at right angles to the scale's direction and is related to either the width or height of the scale. `map` is the structure in which the value and extent, as well as other information is returned.

```
Boolean
    XrtGaUnmap(scale, value, extent, map)
                XrtGaObject    scale;
                double         value;
                double         extent;
                XrtGaMapResult *map;
```

B

Data Types

This appendix lists the XRT/gauge data types: enumerations and structures. They are listed in alphabetical order within their respective sections.

B.1 Enumerations

XrtGaAlignment

Enumeration used by the Counter object with the `XmNxrtGaResetAlignment` resource to specify the alignment of the reset icon (`XRTGA_ALIGNMENT_CENTER` can not be used in this case), and by the State and Text objects with the `XmNxrtGaAlignment` resource to specify the alignment of the text associated with the object:

```
XRTGA_ALIGNMENT_BEGINNING
XRTGA_ALIGNMENT_CENTER
XRTGA_ALIGNMENT_END
```

XrtGaBorder

Enumeration used by all XRT/gauge widgets, the Counter object, and the Text object with the `XmNxrtGaBorder` resource to specify the border type:

```
XRTGA_BORDER_NONE
XRTGA_BORDER_PLAIN
XRTGA_BORDER_3D_IN
XRTGA_BORDER_3D_OUT
XRTGA_BORDER_ETCHED_IN
XRTGA_BORDER_ETCHED_OUT
XRTGA_BORDER_FRAME_IN
XRTGA_BORDER_FRAME_OUT
XRTGA_BORDER_BEVEL
XRTGA_BORDER_SHADOW
```

XrtGaCalcMode

Enumeration used by the Percent widget with the `XmNxrtGaCalcMode` resource to specify how the data is displayed. In percent mode, the percentage value of each item in the list is used, that is, $\frac{(value - min)}{(max - min)} \times 100$. In ratio mode, the ratio of the

item to the sum of all items in the list is used:

```
XRTGA_CALC_MODE_PERCENT
XRTGA_CALC_MODE_RATIO
```

XrtGaDirection

Enumeration used by the LED Icon widget, the Circular Scale object, and the Linear Scale object with the `XmNxrtGaDirection` resource to specify the direction in which the widget/object should progress:

```
XRTGA_DIRECTION_FORWARD
XRTGA_DIRECTION_BACKWARD
```

XrtGaDisplayMode

Enumeration used by the LEDIcon widget with the `XmNxrtGaDisplayMode` resource to specify the display mode used. In magnitude mode, the magnitude of the `XmNxrtGaValue` resource determines how many icons are turned on. In binary mode, the binary representation of the the value determines which icons are turned on. For example, if the value is 9, its binary representation is 1001, which means that icons 0 and 3 are turned on.

```
XRTGA_DISPLAY_MODE_MAGNITUDE
XRTGA_DISPLAY_MODE_BINARY
```

XrtGaFillPattern

Enumeration used to specify the type of fill pattern:

```
XRTGA_FPAT_NONE = 1
XRTGA_FPAT_SOLID
XRTGA_FPAT_25_PERCENT
XRTGA_FPAT_50_PERCENT
XRTGA_FPAT_75_PERCENT
XRTGA_FPAT_HORIZ_STRIPE
XRTGA_FPAT_VERT_STRIPE
XRTGA_FPAT_45_STRIPE
XRTGA_FPAT_135_STRIPE
XRTGA_FPAT_DIAG_HATCHED
XRTGA_FPAT_CROSS_HATCHED
```

XrtGaImageLayout

Enumeration used by all XRT/gauge widgets and the Counter, Frame, Range, and Text objects with the `XmNxrtGaImageLayout` resource to specify the positioning of the background image:

```
XRTGA_IMAGE_LAYOUT_CENTERED
XRTGA_IMAGE_LAYOUT_TILED
```

XrtGaIndicatorType

Enumeration used by the Indicator Object with the `XmNxrtGaIndicatorType` resource to specify the type of indicator to use:

```
XRTGA_INDICATOR_TYPE_RECTANGLE
XRTGA_INDICATOR_TYPE_CIRCLE
XRTGA_INDICATOR_TYPE_TRIANGLE
XRTGA_INDICATOR_TYPE_POINTER
XRTGA_INDICATOR_TYPE_IMAGE
```

XrtGaInteractionType

Enumeration used by the LEDIcon widget and the Indicator, Needle, and Switch objects with the `XmNxrtGaInteractionType` resource to specify the type of mouse interaction with the widget or object.

Objects: the possible values are `NONE`, `RELATIVE_TO_MOUSE`, which means that the object is always the same distance from the mouse as when the interaction started, and `SNAP_TO_MOUSE`, which means that the object is moved directly to the mouse as soon as the interaction starts.

LEDIcon widget: the values act like a `Boolean` value - there is no mouse interaction when the resource is set to `NONE`, and there is mouse interaction when the resource is set to either of the latter two values. The type of interaction depends on the value of `XmNxrtGaDisplayMode`:

```
XRTGA_INTERACTION_TYPE_NONE
XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE
XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE
```

XrtGaLinePattern

Enumeration used by the State object with the `XmNxrtGaLinePattern` resource to specify the style of line:

```
XRTGA_LPAT_NONE
XRTGA_LPAT_SOLID
XRTGA_LPAT_LONG_DASH
XRTGA_LPAT_DOTTED
XRTGA_LPAT_SHORT_DASH
XRTGA_LPAT_LSL_DASH
XRTGA_LPAT_DASH_DOT
```

XrtGaMarkerType

Enumeration used by the Marker object with the `XmNxrtGaMarkerType` resource to specify the type of marker:

```
XRTGA_MARKER_TYPE_CIRCLE
XRTGA_MARKER_TYPE_TRIANGLE
XRTGA_MARKER_TYPE_ARROW
XRTGA_MARKER_TYPE_IMAGE
```

XrtGaNeedleType

Enumeration used by the Needle object with the `XmNxrtGaNeedleType` resource to specify the type of needle:

```
XRTGA_NEEDLE_TYPE_POINTER
XRTGA_NEEDLE_TYPE_TRIANGLE
XRTGA_NEEDLE_TYPE_ARROW
```

XrtGaOffsetType

Enumeration used by the Circular Dial widget with the `XmNxrtGaOffsetType` resource to specify the type of string offset used for each state of a discrete switch. This only applies when the `XmNxrtGaOffsetUseDefault` resource of a State object is

set to True (this allows the string offsets to be automatically controlled by the parent Circular Dial widget):

```
XRTGA_OFFSET_TYPE_AUTOMATIC
XRTGA_OFFSET_TYPE_CIRCULAR
XRTGA_OFFSET_TYPE_HORIZONTAL
XRTGA_OFFSET_TYPE_VERTICAL
```

XrtGaReason

Used in most callbacks to fill in the reason structure member.

```
XRTGA_REASON_VALUE_CHANGE_BEGIN = 100
XRTGA_REASON_VALUE_CHANGE_END
XRTGA_REASON_INTERACTION_START
XRTGA_REASON_INTERACTION_MOVE
XRTGA_REASON_INTERACTION_END
XRTGA_REASON_RESET_COUNTER_BEGIN
XRTGA_REASON_RESET_COUNTER_END
XRTGA_REASON_MAP
XRTGA_REASON_EXPOSE
XRTGA_REASON_RESIZE
XRTGA_REASON_ZOOM
XRTGA_REASON_TRANSLATE
```

XrtGaShape

Enumeration used by the Linear Dial widget, and the Frame, Icon, and Range objects with the `XmNxrtGaShape` resource to specify the shape displayed.

Linear Dial widget: The shape of the *on*/*off* range. Valid values are `RECTANGLE`, `CIRCLE`, `THERMOMETER`, and `TANK`.

Frame object: The shape of the frame. Valid values are `RECTANGLE` , `CIRCLE` , and `IMAGE` .

Icon object: The values can be `RECTANGLE` or `CIRCLE`, in which case the *on* and *off* color are used to fill the specific shape, or `IMAGE` in which case a specified image is used. The size of the icon is determined by the parent.

Range object: This resource is ignored if the associated scale is circular (in this case a circular band or annular region is always drawn). If not, any of the five values below may be used:

```
XRTGA_SHAPE_RECTANGLE
XRTGA_SHAPE_CIRCLE
XRTGA_SHAPE_THERMOMETER
XRTGA_SHAPE_TANK
XRTGA_SHAPE_IMAGE
```

XrtGaSizingPolicy

Enumeration used by the LEDIcon widget with the `XmNxrtGaSizingPolicy` resource to control the widget size vs icon size. Valid values are `WIDGET`, `ICON`, `BY_POSITION`, and `CENTER`.

`WIDGET`: the widget's width and height determine the size of the icons and any attempt by the end user to set the icon width and height is ignored.

ICON: the icon's height and width determine the size of the widget (if possible), and any user attempts to control the widget's size will be ignored.

BY_POSITION: the XmNxrtGaXOrigin and XmNxrtGaYOrigin resources set the location of the group of LED icons, and the XmNxrtGaIconHeight and XmNxrtGaIconWidth resources determine the size of the icons. The icons are clipped if the widget does not have enough space to display all of them.

CENTER: the XmNxrtGaIconHeight and XmNxrtGaIconWidth resources determine the size of the icons, and the group of LED icons is centered within the widget. The icons are clipped if the widget does not have enough space to display all of them.

```
XRTGA_SIZING_POLICY_WIDGET
XRTGA_SIZING_POLICY_ICON
XRTGA_SIZING_POLICY_BY_POSITION
XRTGA_SIZING_POLICY_CENTER
```

XrtGaStringMode

Enumeration used by the Text object with the XmNxrtGaStringMode resource. If STRING is chosen, then a normal string is used and is pointed to by the XmNxrtGaString resource. If XM_STRING is chosen, then an XmString is used and is pointed to by the XmNxrtGaXmString resource:

```
XRTGA_STRING_MODE_STRING
XRTGA_STRING_MODE_XM_STRING
```

XrtGaSwitchType

Enumeration used by the Switch object with the XmNxrtGaSwitchType resource to specify the switch's type:

```
XRTGA_SWITCH_TYPE_CIRCLE
XRTGA_SWITCH_TYPE_POINTER
XRTGA_SWITCH_TYPE_ARROW
XRTGA_SWITCH_TYPE_FAUCET
XRTGA_SWITCH_TYPE_IMAGE
```

XrtGaTickType

Enumeration used by the Tick object with the XmNxrtGaTickType resource to specify the type of tick to be used:

```
XRTGA_TICK_TYPE_RECTANGLE
XRTGA_TICK_TYPE_CIRCLE
XRTGA_TICK_TYPE_DIAMOND
XRTGA_TICK_TYPE_TRIANGLE
```

XrtGaValueType

Enumeration used by the LEDNumerical widget with the XmNxrtGaValueType resource to specify the type of value to display. If NUMERIC is chosen, then the value is obtained from the resource XmNxrtGaValue. If ALPHANUMERIC is chosen, then the value is obtained from the resource XmNxrtGaText.

```
XRTGA_VALUE_TYPE_NUMERIC
XRTGA_VALUE_TYPE_ALPHANUMERIC
```

B.2 Structures

XrtGaCallbackStruct

Structure used by all XRT/gauge widgets to define the information passed to the Expose and Resize callbacks:

```
typedef struct {
    int              reason;        /* read only */
    XEvent          *event;         /* read only */
    Window           window;        /* read only */
    Region           region;        /* read only */
    int              change;        /* read only */
} XrtGaCallbackStruct;
```

XrtGaDrawData

Structure used by the Indicator, Marker, Needle, and Switch objects to pass the information to `XrtGaDrawFunc` as a `user_data`:

```
typedef struct {
    Position        x;        /* X position of drawing origin */
    Position        y;        /* Y position of drawing origin */
    Dimension       width;    /* Width of draw object */
    Dimension       height;   /* Height of draw object */
    double          angle;    /* Angle at which
                                 the object is to be drawn */
    Position        x_offset; /* x distance from (x, y) */
    Position        y_offset; /* y distance from (x, y) */
} XrtGaDrawData;
```

XrtGaInteractionCallbackStruct

Structure passed to an object's `XmNxrtGaInteractionCallback` resource:

```
typedef struct {
    int         reason;     /* read only */
    XEvent      *event;     /* read only */
    Boolean     doit;       /* read/write when reason is not
                                 XRTGA_REASON_INTERACTION_END. */
} XrtGaInteractionCallbackStruct;
```

XrtGaMapCallbackStruct

Structure used by the Circular Scale and Linear Scale objects that is passed to the object's `XmNxrtGaMapCallback` resource:

```
typedef struct {
    int         reason;     /* read only */
    XEvent      *event;     /* read only */
    String      *params;    /* read only */
    Cardinal    nparams;    /* read only */
    XrtGaObject scale;      /* read only */
    XrtGaMapResult map;     /* read only */
} XrtGaMapCallbackStruct;
```

XrtGaMapResult

Structure used by the Circular Scale and Linear Scale objects that is filled in when `XrtGaMap()` is called:

```
typedef struct {
    Position    pix_x;       /* X pixel position */
    Position    pix_y;       /* Y pixel position */
    double      value;       /* Value of (pix_x, pix_y) in scale */
    double      extent;      /* Extent along radius, width, height */
    double      angle;       /* Angle for circular scales */
    Position    x_offset;    /* X distance from scale origin */
    Position    y_offset;    /* Y distance from scale origin */
} XrtGaMapResult;
```

XrtGaPolyDrawStruct

Structure used by the Switch and Tick objects that is passed to `XrtGaDrawPolyTransform()` and should contain all the necessary information for that function:

```
typedef struct {
    int         operations; /* Bitwise OR of possible XRTGA_* ops */
    Position    x;          /* X position to translate to. */
    Position    y;          /* Y position to translate to. */
    Dimension   width;      /* Width to scale to. */
    Dimension   height;     /* Height to scale to. */
    double      angle;      /* Angle to rotate to (0 degrees is East
                           and angles increase counterclockwise.)*/
    XPoint      *shape;     /* List of points */
    int         npoints;    /* Number of points in "shape". */
} XrtGaPolyDrawStruct;
```

XrtGaTransformCallbackStruct

Structure used by the Circular Scale and Linear Scale objects that is passed to the object's `XmNxrtGaTransformCallback`. If any of the parameters have the value -1, it means that that value has not changed and will not change:

```
typedef struct {
    int         reason;     /* read only, #XrtGaReason */
    XEvent      *event;     /* read only */
    String      *params;    /* read only */
    Cardinal    nparams;    /* read only */
    Dimension   new_size;   /* New radius size for circular
                               scales */
    Dimension   new_height; /* New height for linear scales */
    Dimension   new_width;  /* New width for linear scales */
    Position    new_x;      /* New X location for both linear and
                               circular scales */
    Position    new_y;      /* New Y location for both linear and
                               circular scales */
    Boolean     doit;       /* Default: True. User sets to False
                               to disallow the operation */
} XrtGaTransformCallbackStruct;
```

XrtGaUserDrawInfo

Structure that is returned from `XrtGaGetUserDrawInfo()`:

```
typedef struct {
    Display     *display;      /* Display of this widget/object */
    int         screen;        /* Screen number */
    Window      window;        /* Window ID */
    Pixmap      pixmap;        /* Pixmap used for double buffering
                               (may be \c XmUNSPECIFIED_PIXMAP). */
    Colormap    cmap;          /* Colormap assigned to this
                                  widget or object */
    VisualID    vid;           /* Visual ID to be used. */
    int         visual_class;  /* VisualClass (PseudoColor,
                                  TrueColor, and so on.). */
    Dimension   width;         /* Widget window's width */
    Dimension   height;        /* Widget window's height */
    int         depth;         /* Widget window's depth */
    Boolean     double_buffer; /* XmNxrtGaDoubleBuffer */
} XrtGaUserDrawInfo;
```

XrtGaValueChangedCallbackStruct

Structure used by the LEDIcon and LEDNumerical widgets, and the Counter, Indicator, Item, Needle, and Switch objects, that is passed to the widget or object's `XmNxrtGaValueChangedCallback` resource:

```
typedef struct {
    int         reason;     /* read only */
    XEvent      *event;     /* read only (will be NULL for
                                programatic changes) */
    double      value;      /* read/write when reason is
                               #XRTGA_REASON_VALUE_CHANGE_BEGIN. */
    Boolean     doit;       /* read/write when reason is
                               #XRTGA_REASON_VALUE_CHANGE_BEGIN. */
} XrtGaValueChangedCallbackStruct;
```

C

Resource Types and Classes

This appendix lists the resource type and resource class for each XRT/gauge resource.

C.1 Center Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaCircularScale	XmCXrtGaCircularScale	XmRXrtGaCircularScale
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaRadius	XmCXrtGaRadius	XmRXrtGaDouble

C.2 Circular Dial Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaCircularScale	XmCXrtGaCircularScale	XmRXrtGaCircularScale
XmNxrtGaCounterList	XmCXrtGaCounterList	XmRXrtGaCounterList
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaExposeCallback	XmCCallback	XmRCallback
XmNxrtGaFrame	XmCXrtGaFrame	XmRXrtGaFrame

Name	Class	Type
XmNxrtGaHighlightColor	XmCXrtGaHighlightColor	XmRPixel
XmNxrtGaHighlightSelectedState	XmCXrtGaHighlightSelectedState	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaInteractionCallback	XmCCallback	XmRCallback
XmNxrtGaLineInnerRadius	XmCXrtGaLineRadius	XmRXrtGaDouble
XmNxrtGaLineOuterRadius	XmCXrtGaLineRadius	XmRXrtGaDouble
XmNxrtGaMapCallback	XmCCallback	XmRCallback
XmNxrtGaOffsetDistance	XmCXrtGaOffsetDistance	XmRXrtGaDouble
XmNxrtGaOffsetType	XmCXrtGaOffsetType	XmRXrtGaOffsetType
XmNxrtGaMarker	XmCXrtGaMarker	XmRXrtGaMarker
XmNxrtGaRangeList	XmCXrtGaRangeList	XmRXrtGaRangeList
XmNxrtGaRenderList	XmCXrtGaRenderList	XmRXrtGaRenderList
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaStateList	XmCXrtGaStateList	XmRXrtGaStateList
XmNxrtGaSwitch	XmCXrtGaSwitch	XmRXrtGaSwitch
XmNxrtGaTextList	XmCXrtGaTextList	XmRXrtGaTextList
XmNxrtGaTickList	XmCXrtGaTickList	XmRXrtGaTickList
XmNxrtGaUseRenderList	XmCXrtGaUseRenderList	XmRBoolean

C.3 Circular Gauge Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaCenterList	XmCXrtGaCenterList	XmRXrtGaCenterList
XmNxrtGaCircularScaleList	XmCXrtGaCircularScaleList	XmRXrtGaCircularScaleList
XmNxrtGaCounterList	XmCXrtGaCounterList	XmRXrtGaCounterList
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaExposeCallback	XmCCallback	XmRCallback
XmNxrtGaFrame	XmCXrtGaFrame	XmRXrtGaFrame

Name	Class	Type
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaInteractionCallback	XmCCallback	XmRCallback
XmNxrtGaMapCallback	XmCCallback	XmRCallback
XmNxrtGaNeedleList	XmCXrtGaNeedleList	XmRXrtGaNeedleList
XmNxrtGaRangeList	XmCXrtGaRangeList	XmRXrtGaRangeList
XmNxrtGaRenderList	XmCXrtGaRenderList	XmRXrtGaRenderList
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaTextList	XmCXrtGaTextList	XmRXrtGaTextList
XmNxrtGaTickList	XmCXrtGaTickList	XmRXrtGaTickList
XmNxrtGaUseRenderList	XmCXrtGaUseRenderList	XmRBoolean

C.4 Circular Scale Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaAllowTransform	XmCXrtGaAllowTransform	XmRBoolean
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDirection	XmCXrtGaDirection	XmRXrtGaDirection
XmNxrtGaMax	XmCXrtGaLimits	XmRXrtGaDouble
XmNxrtGaMin	XmCXrtGaLimits	XmRXrtGaDouble
XmNxrtGaRadiusSize	XmCXrtGaRadiusSize	XmRDimension
XmNxrtGaStartAngle	XmCXrtGaAngle	XmRXrtGaDouble
XmNxrtGaStopAngle	XmCXrtGaAngle	XmRXrtGaDouble
XmNxrtGaTransformCallback	XmCCallback	XmRCallback
XmNxrtGaXOrigin	XmCXrtGaPosition	XmRPosition
XmNxrtGaYOrigin	XmCXrtGaPosition	XmRPosition

C.5 Counter Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNfontList	XmCFontList	XmRFontList
XmNheight	XmCHeight	XmRDimension
XmNwidth	XmCWidth	XmRDimension
XmNx	XmCX	XmRPosition
XmNy	XmCY	XmRPosition
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaBorderWidth	XmCXrtGaDimension	XmRDimension
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDecimalPlaces	XmCXrtGaDecimalPlaces	XmRInt
XmNxrtGaDigitBackgroundColor	XmCXrtGaColor	XmRPixel
XmNxrtGaDigitForegroundColor	XmCXrtGaColor	XmRPixel
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaNumDigits	XmCXrtGaNumDigits	XmRInt
XmNxrtGaResetAlignment	XmCXrtGaResetAlignment	XmRXrtGaAlignment
XmNxrtGaResetBackgroundColor	XmCBackground	XmRPixel
XmNxrtGaResetHeight	XmCHeight	XmRDimension
XmNxrtGaResetImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaResetOffset	XmCOffset	XmRDimension
XmNxrtGaResetShape	XmCXrtGaShape	XmRXrtGaShape
XmNxrtGaResetShow	XmCXrtGaResetShow	XmRBoolean
XmNxrtGaResetWidth	XmCWidth	XmRDimension
XmNxrtGaSpacingHorizontal	XmCXrtGaSpacing	XmRDimension
XmNxrtGaSpacingVertical	XmCXrtGaSpacing	XmRDimension
XmNxrtGaTransitionRate	XmCXrtGaTransition	XmRInt
XmNxrtGaTransitionSteps	XmCXrtGaTransition	XmRInt
XmNxrtGaUseTransition	XmCXrtGaUseTransition	XmRBoolean
XmNxrtGaValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback

C.6 Frame Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaBottomMargin	XmCXrtGaMargin	XmRInt
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaLeftMargin	XmCXrtGaMargin	XmRInt
XmNxrtGaRightMargin	XmCXrtGaMargin	XmRInt
XmNxrtGaShape	XmCXrtGaShape	XmRXrtGaShape
XmNxrtGaTopMargin	XmCXrtGaMargin	XmRInt

C.7 Icon Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaOffColor	XmCXrtGaColor	XmRPixel
XmNxrtGaOffImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaOnColor	XmCXrtGaColor	XmRPixel
XmNxrtGaOnImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaShape	XmCXrtGaShape	XmRXrtGaShape

C.8 Indicator Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNwidth	XmCWidth	XmRDimension
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDrawFunc	XmCXrtGaDrawFunc	XmRFunction
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage

Name	Class	Type
XmNxrtGaImageRotate	XmCXrtGaImageRotate	XmRBoolean
XmNxrtGaIndicatorType	XmCXrtGaIndicatorType	XmRXrtGaIndicatorType
XmNxrtGaInnerExtent	XmCXrtGaExtent	XmRXrtGaDouble
XmNxrtGaInteractionType	XmCXrtGaInteractionType	XmRXrtGaInteractionType
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaLinearScale	XmCXrtGaLinearScale	XmRXrtGaLinearScale
XmNxrtGaOuterExtent	XmCXrtGaExtent	XmRXrtGaDouble
XmNxrtGaValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback

C.9 Item Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaLabelExtent	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback

C.10 LEDIcon Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDecayRate	XmCXrtGaDecayRate	XmRInt
XmNxrtGaDirection	XmCXrtGaDirection	XmRXrtGaDirection
XmNxrtGaDisplayMode	XmCXrtGaDisplayMode	XmRXrtGaDisplayMode
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaExposeCallback	XmCCallback	XmRCallback

Name	Class	Type
XmNxrtGaIconHeight	XmCXrtGaDimension	XmRDimension
XmNxrtGaIconList	XmCXrtGaIconList	XmRXrtGaIconList
XmNxrtGaIconSpacing	XmCXrtGaDimension	XmRDimension
XmNxrtGaIconWidth	XmCXrtGaDimension	XmRDimension
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaInteractionCallback	XmCCallback	XmRCallback
XmNxrtGaInteractionType	XmCXrtGaInteractionType	XmRXrtGaInteractionType
XmNxrtGaLEDStartValue	XmCXrtGaIntValue	XmRInt
XmNxrtGaLEDValue	XmCXrtGaIntValue	XmRInt
XmNxrtGaMaxIcon	XmCXrtGaMaxIcon	XmRBoolean
XmNxrtGaOrientation	XmCXrtGaOrientation	XmROrientation
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaSizingPolicy	XmCXrtGaSizingPolicy	XmRXrtGaSizingPolicy
XmNxrtGaSpacingHorizontal	XmCXrtGaSpacing	XmRDimension
XmNxrtGaSpacingVertical	XmCXrtGaSpacing	XmRDimension
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback
XmNxrtGaXOrigin	XmCXrtGaPosition	XmRPosition
XmNxrtGaYOrigin	XmCXrtGaPosition	XmRPosition

C.11 LEDNumerical Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaColorMixPercentage	XmCXrtGaColorMixPercentage	XmRInt
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDecimalPlaces	XmCXrtGaDecimalPlaces	XmRInt
XmNxrtGaDecimalPosition	XmCXrtGaDecimalPosition	XmRPosition
XmNxrtGaDecimalSize	XmCXrtGaDecimalSize	XmRDimension
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaExposeCallback	XmCCallback	XmRCallback

Name	Class	Type
XmNxrtGaFixedDecimal	XmCXrtGaFixedDecimal	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaLeadingOne	XmCXrtGaLeadingOne	XmRBoolean
XmNxrtGaLeadingPlusMinus	XmCXrtGaLeadingPlusMinus	XmRBoolean
XmNxrtGaLeadingZeros	XmCXrtGaLeadingZeros	XmRBoolean
XmNxrtGaNumDigits	XmCXrtGaNumDigits	XmRInt
XmNxrtGaOffColor	XmCXrtGaColor	XmRPixel
XmNxrtGaOnColor	XmCXrtGaColor	XmRPixel
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaSegmentSpacing	XmCXrtGaSpacing	XmRDimension
XmNxrtGaSegmentWidth	XmCXrtGaDimension	XmRDimension
XmNxrtGaSlopeOffset	XmCXrtGaSlopeOffset	XmRPosition
XmNxrtGaSpacingHorizontal	XmCXrtGaSpacing	XmRDimension
XmNxrtGaSpacingVertical	XmCXrtGaSpacing	XmRDimension
XmNxrtGaText	XmCXrtGaText	XmRString
XmNxrtGaValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback
XmNxrtGaValueType	XmCXrtGaValueType	XmRXrtGaValueType

C.12 Linear Dial Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaDrawFunc	XmCXrtGaDrawFunc	XmRFunction
XmNxrtGaExposeCallback	XmCCallback	XmRCallback
XmNxrtGaFrame	XmCXrtGaFrame	XmRXrtGaFrame
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout

Name	Class	Type
XmNxrtGaIndicator	XmCXrtGaIndicator	XmRXrtGaIndicator
XmNxrtGaInteractionCallback	XmCCallback	XmRCallback
XmNxrtGaLinearScale	XmCXrtGaLinearScale	XmRXrtGaLinearScale
XmNxrtGaMapCallback	XmCCallback	XmRCallback
XmNxrtGaOffColor	XmCXrtGaColor	XmRPixel
XmNxrtGaOffImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaOnColor	XmCXrtGaColor	XmRPixel
XmNxrtGaOnImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaRenderList	XmCXrtGaRenderList	XmRXrtGaRenderList
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaShape	XmCXrtGaShape	XmRXrtGaShape
XmNxrtGaTextList	XmCXrtGaTextList	XmRXrtGaTextList
XmNxrtGaTickList	XmCXrtGaTickList	XmRXrtGaTickList
XmNxrtGaUseRenderList	XmCXrtGaUseRenderList	XmRBoolean

C.13 Linear Gauge Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaExposeCallback	XmCCallback	XmRCallback
XmNxrtGaFrame	XmCXrtGaFrame	XmRXrtGaFrame
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaIndicatorList	XmCXrtGaIndicatorList	XmRXrtGaIndicatorList
XmNxrtGaInteractionCallback	XmCCallback	XmRCallback
XmNxrtGaLinearScaleList	XmCXrtGaLinearScaleList	XmRXrtGaLinearScaleList
XmNxrtGaMapCallback	XmCCallback	XmRCallback
XmNxrtGaRangeList	XmCXrtGaRangeList	XmRXrtGaRangeList
XmNxrtGaRenderList	XmCXrtGaRenderList	XmRXrtGaRenderList

Name	Class	Type
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaTextList	XmCXrtGaTextList	XmRXrtGaTextList
XmNxrtGaTickList	XmCXrtGaTickList	XmRXrtGaTickList
XmNxrtGaUseRenderList	XmCXrtGaUseRenderList	XmRBoolean

C.14 Linear Scale Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNheight	XmCHeight	XmRDimension
XmNwidth	XmCWidth	XmRDimension
XmNx	XmCX	XmRPosition
XmNy	XmCY	XmRPosition
XmNxrtGaAllowTransform	XmCXrtGaAllowTransform	XmRBoolean
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDirection	XmCXrtGaDirection	XmRXrtGaDirection
XmNxrtGaMax	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaMin	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaOrientation	XmCXrtGaOrientation	XmROrientation
XmNxrtGaTransformCallback	XmCCallback	XmRCallback

C.15 Marker Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNwidth	XmCWidth	XmRDimension
XmNxrtGaCircularScale	XmCXrtGaCircularScale	XmRXrtGaCircularScale
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDrawFunc	XmCXrtGaDrawFunc	XmRFunction
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage

Name	Class	Type
XmNxrtGaImageRotate	XmCXrtGaImageRotate	XmRBoolean
XmNxrtGaInnerRadius	XmCXrtGaInnerRadius	XmRXrtGaDouble
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaMarkerType	XmCXrtGaMarkerType	XmRXrtGaMarkerType
XmNxrtGaOuterRadius	XmCXrtGaOuterRadius	XmRXrtGaDouble

C.16 Needle Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNwidth	XmCWidth	XmRDimension
XmNxrtGaCircularScale	XmCXrtGaCircularScale	XmRXrtGaCircularScale
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDrawFunc	XmCXrtGaDrawFunc	XmRFunction
XmNxrtGaInteractionType	XmCXrtGaInteractionType	XmRXrtGaInteractionType
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaLength	XmCXrtGaLength	XmRXrtGaDouble
XmNxrtGaNeedleType	XmCXrtGaNeedleType	XmRXrtGaNeedleType
XmNxrtGaValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback

C.17 Percent Widget

Name	Class	Type
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaCalcMode	XmCXrtGaCalcMode	XmRXrtGaCalcMode
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDoubleBuffer	XmCXrtGaDoubleBuffer	XmRBoolean
XmNxrtGaExposeCallback	XmCCallback	XmRCallback
XmNxrtGaFrame	XmCXrtGaFrame	XmRXrtGaFrame
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage

Name	Class	Type
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaItemList	XmCXrtGaItemList	XmRXrtGaItemList
XmNxrtGaMapCallback	XmCCallback	XmRCallback
XmNxrtGaRenderList	XmCXrtGaRenderList	XmRXrtGaRenderList
XmNxrtGaRepaint	XmCXrtGaRepaint	XmRBoolean
XmNxrtGaResizeCallback	XmCCallback	XmRCallback
XmNxrtGaScale	XmCXrtGaScale	XmRXrtGaScale
XmNxrtGaTextList	XmCXrtGaTextList	XmRXrtGaTextList
XmNxrtGaUseRenderList	XmCXrtGaUseRenderList	XmRBoolean

C.18 Range Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaInnerExtent	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaOffset	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaOuterExtent	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaScale	XmCXrtGaScale	XmRXrtGaScale
XmNxrtGaShape	XmCXrtGaShape	XmRXrtGaShape
XmNxrtGaStartValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaStopValue	XmCXrtGaDoubleValue	XmRXrtGaDouble

C.19 State Object

Name	Class	Type
XmNfontList	XmCFontList	XmRFontList
XmNforeground	XmCForeground	XmRPixel

Name	Class	Type
XmNxrtGaAlignment	XmCXrtGaAlignment	XmRXrtGaAlignment
XmNxrtGaAlignmentUseDefault	XmCXrtGaAlignmentUseDefault	XmRBoolean
XmNxrtGaAngle	XmCXrtGaAngle	XmRXrtGaDouble
XmNxrtGaAngleUseDefault	XmCXrtGaAngleUseDefault	XmRBoolean
XmNxrtGaCircularScale	XmCXrtGaCircularScale	XmRXrtGaCircularScale
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaLineDisplay	XmCXrtGaLineDisplay	XmRBoolean
XmNxrtGaLineInnerRadius	XmCXrtGaLineRadius	XmRXrtGaDouble
XmNxrtGaLineInnerRadiusUseDefault	XmCXrtGaLineRadiusUseDefault	XmRBoolean
XmNxrtGaLineOuterRadius	XmCXrtGaLineRadius	XmRXrtGaDouble
XmNxrtGaLineOuterRadiusUseDefault	XmCXrtGaLineRadiusUseDefault	XmRBoolean
XmNxrtGaLinePattern	XmCXrtGaLinePattern	XmRXrtGaLinePattern
XmNxrtGaLineThickness	XmCXrtGaLineThickness	XmRInt
XmNxrtGaOffsetUseDefault	XmCXrtGaOffsetUseDefault	XmRBoolean
XmNxrtGaOffsetX	XmCXrtGaOffsetX	XmRPosition
XmNxrtGaOffsetY	XmCXrtGaOffsetY	XmRPosition
XmNxrtGaText	XmCXrtGaText	XmRString

C.20 Switch Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNxrtGaCircularScale	XmCXrtGaCircularScale	XmRXrtGaCircularScale
XmNxrtGaContinuous	XmCXrtGaContinuous	XmRBoolean
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDrawFunc	XmCXrtGaDrawFunc	XmRFunction
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageRotate	XmCXrtGaImageRotate	XmRBoolean
XmNxrtGaIncrementValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaInteractionType	XmCXrtGaInteractionType	XmRXrtGaInteractionType
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean

Name	Class	Type
XmNxrtGaMultipleTurnEnabled	XmCXrtGaMultipleTurnEnabled	XmRBoolean
XmNxrtGaMultipleTurnValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaRadius	XmCXrtGaRadius	XmRXrtGaDouble
XmNxrtGaSwitchType	XmCXrtGaSwitchType	XmRXrtGaSwitchType
XmNxrtGaValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaValueChangedCallback	XmCCallback	XmRCallback

C.21 Text Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNfontList	XmCFontList	XmRFontList
XmNforeground	XmCForeground	XmRPixel
XmNheight	XmCHeight	XmRDimension
XmNwidth	XmCWidth	XmRDimension
XmNx	XmCX	XmRPosition
XmNy	XmCY	XmRPosition
XmNxrtGaAlignment	XmCXrtGaAlignment	XmRXrtGaAlignment
XmNxrtGaAllowWidgetPositioning	XmCXrtGaAllowWidgetPositioning	XmRBoolean
XmNxrtGaBorder	XmCXrtGaBorder	XmRXrtGaBorder
XmNxrtGaBorderWidth	XmCXrtGaDimension	XmRDimension
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaImage	XmCXrtGaImage	XmRXrtGaImage
XmNxrtGaImageLayout	XmCXrtGaImageLayout	XmRXrtGaImageLayout
XmNxrtGaImageMinimumSize	XmCXrtGaImageMinimumSize	XmRBoolean
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaString	XmCString	XmRString
XmNxrtGaStringMode	XmCXrtGaStringMode	XmRXrtGaStringMode
XmNxrtGaValueObject	XmCXrtGaValueObject	XmRXrtGaValueObject
XmNxrtGaXmString	XmCXmString	XmRXmString

C.22 Tick Object

Name	Class	Type
XmNbackground	XmCBackground	XmRPixel
XmNfontList	XmCFontList	XmRFontList
XmNwidth	XmCWidth	XmRDimension
XmNxrtGaAutomatic	XmCXrtGaAutomatic	XmRBoolean
XmNxrtGaDebug	XmCXrtGaDebug	XmRBoolean
XmNxrtGaDrawLabels	XmCXrtGaDrawLabels	XmRBoolean
XmNxrtGaIncrementValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaInnerExtent	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaIsShowing	XmCXrtGaIsShowing	XmRBoolean
XmNxrtGaLabelExtent	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaLabelFunc	XmCXrtGaFunction	XmRFunction
XmNxrtGaOffset	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaOuterExtent	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaPrecision	XmCXrtGaPrecision	XmRInt
XmNxrtGaPrecisionUseDefault	XmCXrtGaPrecisionUseDefault	XmRBoolean
XmNxrtGaScale	XmCXrtGaScale	XmRXrtGaScale
XmNxrtGaStartValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaStopValue	XmCXrtGaDoubleValue	XmRXrtGaDouble
XmNxrtGaTickType	XmCXrtGaTickType	XmRXrtGaTickType

D

Inherited Resources

This appendix lists all of the inherited resources that each widget in XRT/gauge uses. Listed after the resource name is 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()*).

XmPrimitive Resource Set			
Name	**Type**	**Default**	**Access**
XmNbottomShadowColor	Pixel	dynamic	CSG
XmNbottomShadowPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNforeground	Pixel	dynamic	CSG
XmNhelpCallback	XtCallbackList	NULL	C
XmNhighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	Pixmap	dynamic	CSG
XmNhighlightThickness	Dimension	2	CSG
XmNnavigationType	XmNavigationType	XmNONE	G
XmNshadowThickness	Dimension	2	CSG
XmNtopShadowColor	Pixel	dynamic	CSG
XmNtopShadowPixmap	Pixmap	dynamic	CSG
XmNtraversalOn	Boolean	True	CSG
XmNunitType	unsigned char	dynamic	CSG
XmNuserData	Pointer	NULL	CSG

Core Resource Set			
Name	**Type**	**Default**	**Access**
XmNaccelerators	XtAccelerators	NULL	CSG
XmNancestorSensitive	Boolean	dynamic	G
XmNbackground	Pixel	dynamic	CSG
XmNbackgroundPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNborderColor	Pixel	XtDefaultForeground	CSG
XmNborderPixmap	Pixmap	XmUNSPECIFIED_PIXMAP	CSG
XmNborderWidth	Dimension	0	CSG
XmNcolormap	Colormap	dynamic	CG
XmNdepth	int	dynamic	CG
XmNdestroyCallback	XtCallbackList	NULL	C
XmNheight	Dimension	dynamic	CSG
XmNinitialResourcesPersistent	Boolean	True	C
XmNmappedWhenManaged	Boolean	True	CSG
XmNscreen	Screen *	dynamic	CG
XmNsensitive	Boolean	True	CSG
XmNtranslations	XtTranslations	NULL	CSG
XmNwidth	Dimension	dynamic	CSG
XmNx	Position	0	CSG
XmNy	Position	0	CSG

E

Resource File and GUI Builder Usage

This appendix lists the syntax for setting XRT/gauge resources in resource files using GUI builder tools. It also lists a sample program and the associated resource file. XRT/gauge only defines resource converters for those data types not already defined by Xt and Motif.

All XRT/gauge resources except `UseDefault` resources and object list resources may be defined in a resource file.

E.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 using enumerated types use a consistent naming convention based on the enumerated values. For all resource types, the leading `XRTGA_` is dropped, and any remaining underscores are removed. The resource values can be entered in upper, lower, or mixed case. For example, each of the following resource specifications causes the value `XRTGA_BORDER_SHADOW` to be used for `XmNxrtGaBorder` in a Circular Dial widget:

```
*.XmXrtGaCircularDial.xrtGaBorder: BorderShadow
*.XmXrtGaCircularDial.xrtGaBorder: BORDERSHADOW
*.XmXrtGaCircularDial.xrtGaBorder: bordershadow
```

Images

Resources of type `XmRXrtGaImage` are specified with any valid filename. Filenames are checked for file type and are then converted to the proper image type if it is a default image type or an image type registered by the user. An example:

```
*.XmXrtGaLinearDial.xrtGaOnImage:$XRTHOME/src/icons/smile32.xpm
```

Objects

Resources of type `XmRXrtGaObject` can be specified in a resource file. The objects specified in a resource file must be retrieved by `XtName`. Any object referenced must be created before the object or widget referencing it. For example, if a Range object wants to set its `XmNxrtGaScale` resource in a resource file, it must be created after the scale. The sample code and resource file in the next section demonstrate the correct order of object creation in the sample code, in conjunction with the resource file.

Lists of Objects

Resources of type `XmRXrtGaObjectList` cannot be specified in a resource file.

String Values

Strings may be specified using the standard Motif string specification.

E.2 Example Code and Resource File

The following image is defined by the combination of the sample program and sample resource file below:

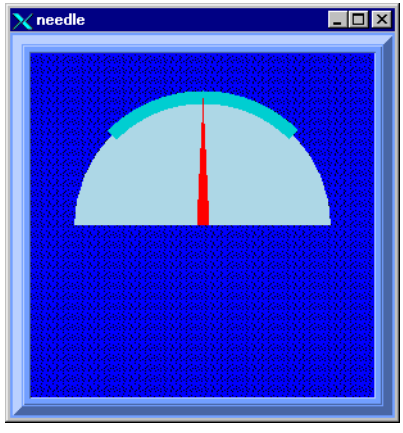


Figure 47 A circular gauge whose resources are set in a resource file

E.2.1 Sample Program

First, here is a bare-bones program that creates a circular gauge widget and puts it in a form:

```
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/XrtGaugeAll.h>
#include "mtgauge.h"

int
main(argc, argv)
int argc;
char *argv[];
{
    XtAppContext          app_context;
    Widget                toplevel, form, cgauge, scale, range, needle;

putenv("XENVIRONMENT=./needle.res");


    /* Initialize Intrinsics */
    toplevel = XtVaAppInitialize(&app_context, "CGauge", NULL, 0,
                                 &argc, argv, NULL, NULL);

    /* Create Form*/
    form = XtVaCreateManagedWidget("form",
                                   xmFormWidgetClass,       toplevel,
                                   NULL);


    /* Create Gauge */
```

```
    cgauge =
        XtVaCreateManagedWidget("circular_gauge",
                                xmXrtGaCircularGaugeWidgetClass, form,
                                NULL);
    /* Create Basic Scale */
    scale = XtVaCreateWidget("scale",
                             xmXrtGaCircularScaleObjectClass,  cgauge,
                             NULL);

    /* Create Range */
    range = XtVaCreateWidget("range",
                             xmXrtGaRangeObjectClass,          cgauge,
                             NULL);

    /* Create Needle */
    needle = XtVaCreateWidget("needle",
                              xmXrtGaNeedleObjectClass,        cgauge,
                              NULL);

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

The resources are set in a resource file. Note that forward references are not allowed, so the scale must be created before the range and the needle.

```
CGauge*.circular_gauge.xrtGaImage:       $XRTHOME/src/icons/shading.xpm
CGauge*.circular_gauge.xrtGaImageLayout:    ImageLayoutTiled
CGauge*.circular_gauge.xrtGaBorder:         BorderFrameOut
CGauge*.circular_gauge.shadowThickness:     14
CGauge*.scale.background:                   lightblue
CGauge*.scale.xrtGaStartAngle:              0.0
CGauge*.scale.xrtGaStopAngle:               180.0
CGauge*.scale.xrtGaMin:                     50.0
CGauge*.scale.xrtGaMax:                     150.0
CGauge*.scale.xrtGaDirection:               DirectionBackward
CGauge*.range.xrtGaScale:                   scale
CGauge*.range.xrtGaInnerExtent:             95%
CGauge*.range.xrtGaOuterExtent:             105%
CGauge*.range.xrtGaStartValue:              75.0
CGauge*.range.xrtGaStopValue:               125.0
CGauge*.range.background:                   DarkTurquoise
CGauge*.needle.background:                  red
CGauge*.needle.width:                       10
CGauge*.needle.xrtGaCircularScale:          scale
CGauge*.needle.xrtGaInteractionType:        InteractionTypeSnapToMouse
CGauge*.needle.xrtGaNeedleType:             NeedleTypeTriang
```

Note that the result of compiling and running the program is shown in Figure 47

F

UIL Arguments

XRT/gauge 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/gauge 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 widget and object in XRT/gauge.

Example programs that use UIL are located in *$XRTHOME/src/gauge/examples/simple_cpp.c* (using *simple_cpp.uil*).

F.1 Center Object Resources

UIL Argument Name	Argument Type
XmNxrtGaCircularScale	widget
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaIsShowing	boolean
XmNxrtGaRadius	float

F.2 Circular Dial Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaFrame	widget
XmNxrtGaHighlightColor	color
XmNxrtGaHighlightSelectedState	boolean
XmNxrtGaImageLayout	integer
XmNxrtGaLineInnerRadius	float
XmNxrtGaLineOuterRadius	float
XmNxrtGaOffsetDistance	float
XmNxrtGaOffsetType	integer
XmNxrtGaMarker	widget
XmNxrtGaRepaint	boolean
XmNxrtGaSwitch	widget
XmNxrtGaUseRenderList	boolean

Reasons
XmNxrtGaInteractionCallback
XmNxrtGaExposeCallback
XmNxrtGaResizeCallback

F.3 Circular Gauge Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaDebug	boolean
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaFrame	widget
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaLineInnerRadius	float

UIL Argument Name	Argument Type
XmNxrtGaLineOuterRadius	float
XmNxrtGaRepaint	boolean
XmNxrtGaUseRenderList	boolean

Reasons
XmNxrtGaExposeCallback
XmNxrtGaInteractionCallback
XmNxrtGaResizeCallback

F.4 Circular Scale Object Resources

UIL Argument Name	Argument Type
XmNxrtGaAllowTransform	boolean
XmNxrtGaDebug	boolean
XmNxrtGaDirection	integer
XmNxrtGaMax	float
XmNxrtGaMin	float
XmNxrtGaRadiusSize	integer
XmNxrtGaStartAngle	float
XmNxrtGaStopAngle	float
XmNxrtGaXOrigin	integer
XmNxrtGaYOrigin	integer

Reasons
XmNxrtGaTransformCallback

F.5 Counter Object Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaBorderWidth	integer
XmNxrtGaDebug	boolean
XmNxrtGaDecimalPlaces	integer
XmNxrtGaDigitBackgroundColor	color
XmNxrtGaDigitForegroundColor	color
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaIsShowing	boolean
XmNxrtGaNumDigits	integer
XmNxrtGaResetAlignment	integer
XmNxrtGaResetHeight	integer
XmNxrtGaResetIcon	widget
XmNxrtGaResetOffset	integer
XmNxrtGaResetShow	boolean
XmNxrtGaResetWidth	integer
XmNxrtGaSpacingHorizontal	integer
XmNxrtGaSpacingVertical	integer
XmNxrtGaTransitionRate	integer
XmNxrtGaTransitionSteps	integer
XmNxrtGaUseTransition	boolean
XmNxrtGaValue	float

Reasons
XmNxrtGaValueChangedCallback

F.6 Frame Object Resources

UIL Argument Name	Argument Type
XmNxrtGaBottomMargin	integer
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaIsShowing	boolean
XmNxrtGaLeftMargin	integer
XmNxrtGaRightMargin	integer
XmNxrtGaShape	integer
XmNxrtGaTopMargin	integer

F.7 Icon Object Resources

UIL Argument Name	Argument Type
XmNxrtGaDebug	boolean
XmNxrtGaOffColor	color
XmNxrtGaOffImage	pixmap
XmNxrtGaOnColor	color
XmNxrtGaOnImage	pixmap
XmNxrtGaShape	integer

F.8 Indicator Object Resources

UIL Argument Name	Argument Type
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageRotate	boolean
XmNxrtGaIndicatorType	integer
XmNxrtGaInnerExtent	float
XmNxrtGaInteractionType	integer
XmNxrtGaIsShowing	boolean

UIL Argument Name	Argument Type
XmNxrtGaLinearScale	widget
XmNxrtGaOuterExtent	float
XmNxrtGaValue	float

Reasons
XmNxrtGaValueChangedCallback

F.9 Item Object Resources

UIL Argument Name	Argument Type
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaIsShowing	boolean
XmNxrtGaLabelExtent	float
XmNxrtGaValue	float

Reasons
XmNxrtGaValueChangedCallback

F.10 LEDIcon Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaDebug	boolean
XmNxrtGaDecayRate	integer
XmNxrtGaDirection	integer
XmNxrtGaDisplayMode	integer
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaIconHeight	integer
XmNxrtGaIconSpacing	integer
XmNxrtGaIconWidth	integer
XmNxrtGaImage	pixmap

UIL Argument Name	Argument Type
XmNxrtGaImageLayout	integer
XmNxrtGaInteractionType	integer
XmNxrtGaLEDStartValue	integer
XmNxrtGaLEDValue	integer
XmNxrtGaMaxIcon	boolean
XmNxrtGaOrientation	integer
XmNxrtGaRepaint	boolean
XmNxrtGaSizingPolicy	integer
XmNxrtGaSpacingHorizontal	integer
XmNxrtGaSpacingVertical	integer
XmNxrtGaXOrigin	integer
XmNxrtGaYOrigin	integer

Reasons
XmNxrtGaExposeCallback
XmNxrtGaInteractionCallback
XmNxrtGaResizeCallback
XmNxrtGaValueChangedCallback

F.11 LEDNumerical Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaColorMixPercentage	integer
XmNxrtGaDebug	boolean
XmNxrtGaDecimalPlaces	integer
XmNxrtGaDecimalPosition	integer
XmNxrtGaDecimalSize	integer
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaFixedDecimal	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaLeadingOne	boolean
XmNxrtGaLeadingPlusMinus	boolean
XmNxrtGaLeadingZeros	boolean
XmNxrtGaNumDigits	integer
XmNxrtGaOffColor	color
XmNxrtGaOnColor	color
XmNxrtGaRepaint	boolean
XmNxrtGaSegmentSpacing	integer
XmNxrtGaSegmentWidth	integer
XmNxrtGaSlopeOffset	integer
XmNxrtGaSpacingHorizontal	integer
XmNxrtGaSpacingVertical	integer
XmNxrtGaText	string
XmNxrtGaValue	float
XmNxrtGaValueType	integer

Reasons
XmNxrtGaExposeCallback
XmNxrtGaResizeCallback
XmNxrtGaValueChangedCallback

F.12 Linear Dial Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaDebug	boolean
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaFrame	widget
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaIndicator	widget
XmNxrtGaLinearScale	widget
XmNxrtGaOffColor	color
XmNxrtGaOffImage	pixmap
XmNxrtGaOnColor	color
XmNxrtGaOnImage	pixmap
XmNxrtGaRepaint	boolean
XmNxrtGaShape	integer
XmNxrtGaUseRenderList	boolean

Reasons
XmNxrtGaExposeCallback
XmNxrtGaInteractionCallback
XmNxrtGaResizeCallback

F.13 Linear Gauge Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaDebug	boolean
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaFrame	widget
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer

UIL Argument Name	Argument Type
XmNxrtGaRepaint	boolean
XmNxrtGaUseRenderList	boolean

Reasons
XmNxrtGaExposeCallback
XmNxrtGaInteractionCallback
XmNxrtGaResizeCallback

F.14 Linear Scale Object Resources

UIL Argument Name	Argument Type
XmNxrtGaAllowTransform	boolean
XmNxrtGaDebug	boolean
XmNxrtGaDirection	integer
XmNxrtGaMax	float
XmNxrtGaMin	float
XmNxrtGaOrientation	integer

Reasons
XmNxrtGaTransformCallback

F.15 Marker Object Resources

UIL Argument Name	Argument Type
XmNxrtGaCircularScale	widget
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageRotate	boolean
XmNxrtGaInnerRadius	float
XmNxrtGaIsShowing	boolean
XmNxrtGaMarkerType	integer
XmNxrtGaOuterRadius	float

F.16 Needle Object Resources

UIL Argument Name	Argument Type
XmNxrtGaCircularScale	widget
XmNxrtGaDebug	boolean
XmNxrtGaInteractionType	integer
XmNxrtGaIsShowing	boolean
XmNxrtGaLength	float
XmNxrtGaNeedleType	integer
XmNxrtGaValue	float

Reasons
XmNxrtGaValueChangedCallback

F.17 Percent Widget Resources

UIL Argument Name	Argument Type
XmNxrtGaBorder	integer
XmNxrtGaCalcMode	integer
XmNxrtGaDebug	boolean
XmNxrtGaDoubleBuffer	boolean
XmNxrtGaFrame	widget
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaRepaint	boolean
XmNxrtGaScale	widget
XmNxrtGaUseRenderList	boolean

Reasons
XmNxrtGaExposeCallback
XmNxrtGaResizeCallback

F.18 Range Object Resources

UIL Argument Name	Argument Type
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaInnerExtent	float
XmNxrtGaIsShowing	boolean
XmNxrtGaOffset	float
XmNxrtGaOuterExtent	float
XmNxrtGaScale	widget
XmNxrtGaShape	integer
XmNxrtGaStartValue	float
XmNxrtGaStopValue	float

F.19 State Object Resources

UIL Argument Name	Argument Type
XmNxrtGaAlignment	integer
XmNxrtGaAlignmentUseDefault	boolean
XmNxrtGaAngle	float
XmNxrtGaAngleUseDefault	boolean
XmNxrtGaCircularScale	widget
XmNxrtGaDebug	boolean
XmNxrtGaIsShowing	boolean
XmNxrtGaLineDisplay	boolean
XmNxrtGaLineInnerRadius	float
XmNxrtGaLineInnerRadiusUseDefault	boolean
XmNxrtGaLineOuterRadius	float
XmNxrtGaLineOuterRadiusUseDefault	boolean
XmNxrtGaLinePattern	integer
XmNxrtGaLineThickness	integer
XmNxrtGaOffsetUseDefault	boolean

UIL Argument Name	Argument Type
XmNxrtGaOffsetX	integer
XmNxrtGaOffsetY	integer
XmNxrtGaText	string

F.20 Switch Object Resources

UIL Argument Name	Argument Type
XmNxrtGaCircularScale	widget
XmNxrtGaContinuous	boolean
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageRotate	boolean
XmNxrtGaIncrementValue	float
XmNxrtGaInteractionType	integer
XmNxrtGaIsShowing	boolean
XmNxrtGaMultipleTurnEnabled	boolean
XmNxrtGaMultipleTurnValue	float
XmNxrtGaRadius	float
XmNxrtGaSwitchType	integer
XmNxrtGaValue	float

Reasons
XmNxrtGaValueChangedCallback

F.21 Text Object Resources

UIL Argument Name	Argument Type
XmNxrtGaAlignment	integer
XmNxrtGaAllowWidgetPositioning	boolean
XmNxrtGaBorder	integer
XmNxrtGaBorderWidth	integer

UIL Argument Name	Argument Type
XmNxrtGaDebug	boolean
XmNxrtGaImage	pixmap
XmNxrtGaImageLayout	integer
XmNxrtGaImageMinimumSize	boolean
XmNxrtGaIsShowing	boolean
XmNxrtGaString	string
XmNxrtGaStringMode	integer
XmNxrtGaValueObject	widget
XmNxrtGaXmString	compound_string

F.22 Tick Object Resources

UIL Argument Name	Argument Type
XmNxrtGaAutomatic	boolean
XmNxrtGaDebug	boolean
XmNxrtGaDrawLabels	boolean
XmNxrtGaIncrementValue	float
XmNxrtGaInnerExtent	float
XmNxrtGaIsShowing	boolean
XmNxrtGaLabelExtent	float
XmNxrtGaOffset	float
XmNxrtGaOuterExtent	float
XmNxrtGaPrecision	integer
XmNxrtGaPrecisionUseDefault	boolean
XmNxrtGaScale	widget
XmNxrtGaStartValue	float
XmNxrtGaStopValue	float
XmNxrtGaTickType	integer

F.23 Truncated Constant Definitions

The following XRT/gauge constant definitions have been truncated to fit into the UIL 31-character limit:

- `XRTGA_INTERACTION_TYPE_NONE` becomes `GA_INTERACT_NONE`
- `XRTGA_INTERACTION_TYPE_RELATIVE_TO_MOUSE` becomes `GA_INTERACT_RELATIVE_TO_MOUSE`
- `XRTGA_INTERACTION_TYPE_SNAP_TO_MOUSE` becomes `GA_INTERACT_SNAP_TO_MOUSE`

G

Sample Code

This appendix provides an overview of the sample code included with XRT/gauge. XRT/gauge 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/gauge programming tasks are covered in the examples and demos.

G.1 Examples

Example programs are located in *$XRTHOME/src/gauge/examples*:

analyzer.c	Displays a spectrum analyzer using a Linear Gauge widget or an LEDIcon widget.
cdial.c	Displays a Circular Dial widget.
clock.c	Displays the time in analog and digital modes
color.c	Choose the RGB values for a color using indicators on a linear gauge
combo_lock.c	A game that asks you to use circular dials to open the combination lock before time runs out
expenses.c	Use the Percent widget to a pie chart of household expenses
flashing.c	Use LEDNumerical to show a VCR's "flashing 12"
image.c	Show on/off images on a linear dial
ldial.c	A linear dial example
led_icon.c	Set various resources on an LEDIcon widget and enable the mouse to turn icons on and off
led_num.c	Set various resources on an LEDNumerical widget

led_num_text.c	Set the number of digits and text in an LEDNumerical widget
lgauge.c	Set various resources on a Linear Gauge widget
needle.c	Set various resources on a Needle object in a Circular Gauge
oven.c	Simulate oven controls with Circular Dials
percent.c	Set various resources on a Percent widget
progress.c	Two Progress widgets, one with a circular scale and one with a linear scale
scoreboard.c	Use LEDNumerical widgets that resembles a baseball scoreboard
switch.c	Demonstrates the different switch types for a Circular Dial
temperature.c	A Linear Gauge whose Range object is shaped like a thermometer
ticks.c	Set various resources on Tick objects

G.2 Demos

Demo applications are located in subdirectories of *$XRTHOME/src/gauge/demos*. There is a directory for each demo.

car.c	Displays a car's instrument cluster.
flight	An application that mimics an airplane's instrument panel.
nim.c	Play the game of NIM using LEDIcon widgets
sampler	A collection of the XRT/gauge examples, grouped by widget.

Index

A

action routines
  ActionStart() 168
  Map() 175
  ResetCounter() 173
  TranslateStart() 170
  TranslateUpdate() 169, 172
  ZoomEnd() 172
  ZoomStart() 170, 172
ActionStart() action 168
alignment resource
  in state object 140
  in text object 155
allow transform resource
  in circular scale object 84
  in linear scale object 118
allow widget positioning resource
  in text object 155
angle resource
  in state object 141
angle use default resource
  in state object 141
ANSI C, compatibility with XRT/gauge 12
authorizing
  shared libraries 11
authorizing a program 11
automatic resource
  in tick object 162

B

background resource
  in center object 79
  in circular dial/gauge widget 28
  in circular scale object 84
  in counter object 90
  in frame object 97
  in icon object 100
  in indicator object 105
  in item object 111
  in LEDIcon widget 51
  in LEDNumerical widget 62
  in linear dial/gauge widget 41
  in linear scale object 118
  in marker object 123
  in needle object 127
  in percent widget 73
  in range object 133
  in switch object 148
  in text object 155
  in tick object 162
batching resource updates 14
border resource
  in circular dial/gauge widget 28
  in counter object 91
  in LEDIcon widget 51
  in LEDNumerical widget 62
  in linear dial/gauge widget 41
  in percent widget 73
  in text object 156
border width resource
  in counter object 91
  in text object 156
borders
  in linear dials and gauges 36
bottom margin resource
  in frame object 97

C

C++
  compatibility with XRT/gauge 12
  using with XRT/gauge 12
calc mode resource
  in percent widget 73
callbacks
  interaction 169
center list resource
  in circular gauge widget 30
center object 77
center object resources
  XmNbackground 79
  XmNxrtGaCircularScale 79
  XmNxrtGaDebug 79
  XmNxrtGaImage 79
  XmNxrtGaIsShowing 79
  XmNxrtGaRadius 79
circular dial widget 1, 17
  example of 25
circular dial widget resources
  XmNxrtGaCircularScale 30
  XmNxrtGaHighlightColor 30
  XmNxrtGaHighlightSelectedState 30
  XmNxrtGaLineInnerRadius 31
  XmNxrtGaLineOuterRadius 31

XmNxrtGaMarker 31
XmNxrtGaOffsetDistance 31
XmNxrtGaOffsetType 31
XmNxrtGaStateList 32
XmNxrtGaSwitch 32
circular dial/gauge widget resources
XmNbackground 28
XmNxrtGaBorder 28
XmNxrtGaCounterList 28
XmNxrtGaDebug 28
XmNxrtGaDoubleBuffer 28
XmNxrtGaExposeCallback 28
XmNxrtGaFrame 28
XmNxrtGaImage 29
XmNxrtGaImageLayout 29
XmNxrtGaInteractionCallback 29
XmNxrtGaMapCallback 29
XmNxrtGaRangeList 29
XmNxrtGaRenderList 29
XmNxrtGaRepaint 29
XmNxrtGaResizeCallback 29
XmNxrtGaTextList 29
XmNxrtGaTickList 30
XmNxrtGaUseRenderList 30
circular gauge widget 1, 17
circular gauge widget resources
XmNxrtGaCenterList 30
XmNxrtGaCircularScaleList 30
XmNxrtGaNeedleList 30
circular scale list resource
in circular gaugel widget 30
circular scale object 81
circular scale object resources
XmNbackground 84
XmNxrtGaAllowTransform 84
XmNxrtGaDebug 84
XmNxrtGaDirection 84
XmNxrtGaMax 84
XmNxrtGaMin 84
XmNxrtGaRadiusSize 84
XmNxrtGaStartAngle 84
XmNxrtGaStopAngle 84
XmNxrtGaTransformCallback 85
XmNxrtGaXOrigin 85
XmNxrtGaYOrigin 85
circular scale resource
in center object 79
in circular dial widget 30
in marker object 123
in needle object 127
in state object 141
in switch object 148
clock displays
in LEDNumerical widget 60
color mix percentage resource
in LEDNumerical widget 62
COLOR_RES 217
colors
setting background colors in linear dials and gauges 36
comments on product 3
compiling and authorizing a program 11
continuous resource
in switch object 148
controlling interactive transform, zoom 170, 172
counter list resource
in circular dial/gauge widget 28
counter object 87
counter object resources
XmNbackground 90
XmNfontList 90
XmNheight 90
XmNwidth 90
XmNx 90
XmNxrtGaBorder 91
XmNxrtGaBorderWidth 91
XmNxrtGaDebug 91
XmNxrtGaDecimalPlaces 91
XmNxrtGaDigitBackgroundColor 91
XmNxrtGaDigitForegroundColor 91
XmNxrtGaImage 91
XmNxrtGaImageLayout 92
XmNxrtGaIsShowing 92
XmNxrtGaNumDigits 92
XmNxrtGaResetAlignment 92
XmNxrtGaResetBackgroundColor 92
XmNxrtGaResetHeight 92
XmNxrtGaResetImage 92
XmNxrtGaResetOffset 92
XmNxrtGaResetShape 92
XmNxrtGaResetShow 92
XmNxrtGaResetWidth 93
XmNxrtGaSpacingHorizontal 93
XmNxrtGaSpacingVertical 93
XmNxrtGaTransitionRate 93
XmNxrtGaTransitionSteps 93
XmNxrtGaUseTransition 93
XmNxrtGaValue 93
XmNxrtGaValueChangedCallback 93
XmNy 91

D

data type listing 221
debug resource
in center object 79
in circular dial/gauge widget 28
in circular scale object 84
in counter object 91
in frame object 97
in icon object 101
in indicator object 105
in item object 111
in LEDIcon widget 51
in LEDNumerical widget 62
in linear dial/gauge widget 41
in linear scale object 118
in marker object 123
in needle object 127

in percent widget 73
in range object 133
in state object 141
in switch object 148
in text object 156
in tick object 163
decay rate resource
in LEDIcon widget 51
decimal places resource
in counter object 91
in LEDNumerical widget 62
decimal position resource
in LEDNumerical widget 62
decimal size resource
in LEDNumerical widget 62
demo programs, discussion of 268
destroy procedure 196
dial, circular, See circular dial widget
dial, linear, See linear dial widget
digit background color resource
in counter object 91
digit foreground color resource
in counter object 91
direction
in a linear scale object 117
of travel 83
direction resource
in circular scale object 84
in LEDIcon widget 51
in linear scale object 118
disabling all user interaction 169
disabling user interaction 171
display mode resource
in LEDIcon widget 51
double buffer resource
in circular dial/gauge widget 28
in LEDIcon widget 51
in LEDNumerical widget 62
in linear dial/gauge widget 41
in percent widget 73
draw function
providing your own 181
draw function resource
in indicator object 105
in marker object 123
in needle object 127
in switch object 148
draw labels resource
in tick object 163
DVAL 217

E

enable multiple turns resource
in switch object 149
enumerations 221
XrtGaAlignment 221
XrtGaBorder 221
XrtGaCalcMode 221
XrtGaDirection 222
XrtGaDisplayMode 222
XrtGaFillPattern 222
XrtGaImageLayout 222
XrtGaIndicatorType 222
XrtGaInteractionType 223
XrtGaLinePattern 223
XrtGaMarkerType 223
XrtGaNeedleType 223
XrtGaOffsetType 223
XrtGaShape 224
XrtGaSizingPolicy 224
XrtGaStringMode 225
XrtGaSwitchType 225
XrtGaTickType 225
XrtGaValueType 225
enums
reason 224
event handling procedures 178
example programs, listing of 267
executable, compiling and authorizing 11
Expose callback 179
expose callback resource
in circular dial/gauge widget 28
in LEDIcon widget 52
in LEDNumerical widget 62
in linear dial/gauge widget 41
in percent widget 73
extent
of an indicator 104
extent, inner, resource
in indicator object 106
in range object 134
in tick object 163
extent, label, resource
in item object 112
in tick object 163
extent, outer, resource
in range object 134
in tick object 164
extent,outer, resource
in indicator object 106

F

fixed decimal resource
in LEDNumerical widget 62
font list resource
in counter object 90
in state object 140
in text object 155
in tick object 162
foreground resource
in state object 140
in text object 155
frame object 95
frame object resources
XmNbackground 97
XmNxrtGaBottomMargin 97

XmNxrtGaDebug 97
XmNxrtGaImage 97
XmNxrtGaImageLayout 97
XmNxrtGaIsShowing 97
XmNxrtGaLeftMargin 97
XmNxrtGaRightMargin 97
XmNxrtGaTopMargin 98
frame resource
in circular dial/gauge widget 28
in linear dial/gauge widget 42
in percent widget 73
functions
map 174

G

gauge, circular, See circular gauge widget
gauge, linear, See linear gauge widget
GUI builder usage 12
GUI builders
resource setting syntax 247

H

height resource
in counter object 90
in linear scale object 118
in text object 155
highlight color resource
in circular dial widget 30
highlight selected state resource
in circular dial widget 30
horizontal spacing resource
in counter object 93
in LEDIcon widget 54
in LEDNumerical widget 64

I

icon height resource
in LEDIcon widget 52
icon list resource
in LEDIcon widget 52
icon object 99
shapes 100
using images 100
icon object resources
XmNbackground 100
XmNxrtGaDebug 101
XmNxrtGaOffColor 101
XmNxrtGaOffImage 101
XmNxrtGaOnColor 101
XmNxrtGaOnImage 101
XmNxrtGaShape 101
icon spacing resource
in LEDIcon widget 52
icon width resource
in LEDIcon widget 52
image
in linear dials and gauges 37
image layout resource
in circular dial/gauge widget 29
in counter object 92
in frame object 97
in LEDIcon widget 52
in LEDNumerical widget 63
in linear dial/gauge widget 42
in percent widget 73
in range object 134
in text object 156
image minimum size resource
in text object 156
image resource
in center object 79
in circular dial/gauge widget 29
in counter object 91
in frame object 97
in indicator object 105
in item object 112
in LEDIcon widget 52
in LEDNumerical widget 63
in linear dial/gauge widget 42
in marker object 123
in percent widget 73
in range object 134
in switch object 149
in text object 156
image rotate resource
in indicator object 105
in marker object 123
in switch object 149
increment value resource
in switch object 149
in tick object 163
indicator extents 104
indicator list resource
in linear gauge widget 43
indicator object 103
indicator object resources
XmNbackground 105
XmNwidth 105
XmNxrtGaDebug 105
XmNxrtGaDrawFunc 105
XmNxrtGaImage 105
XmNxrtGaImageRotate 105
XmNxrtGaIndicatorType 106
XmNxrtGaInnerExtent 106
XmNxrtGaInteractionType 106
XmNxrtGaIsShowing 106
XmNxrtGaLinearScale 106
XmNxrtGaOuterExtent 106
XmNxrtGaValue 106
XmNxrtGaValueChangedCallback 106
indicator resource
in linear dial widget 43
indicator type resource
in indicator object 106

indicators
adding to a linear gauge widget 37
inear dial widget 35
inner extent resource
in indicator object 106
in range object 134
in tick object 163
inner radius resource
in circular dial widget 31
in marker object 123
inner radius, line, resource
in state object 141
interaction callback 169, 170
interaction callback resource
in circular dial/gauge widget 29
in LEDIcon widget 52
in linear dial/gauge widget 42
interaction type resource
in indicator object 106
in LEDIcon widget 52
in needle object 127
in switch object 149
interactions in XRT/gauge 167
internationalization 13
is showing resource
in center object 79
in counter object 92
in frame object 97
in indicator object 106
in item object 112
in marker object 123
in needle object 127
in range object 134
in state object 139, 141
in switch object 149
in text object 156
in tick object 163
item list resource
in percent widget 73
item object 109
item object resources
XmNbackground 111
XmNxrtGaDebug 111
XmNxrtGaImage 112
XmNxrtGaIsShowing 112
XmNxrtGaLabelExtent 112
XmNxrtGaValue 112
XmNxrtGaValueChangedCallback 112
items
in percent widget 67

L

label extent resource
in item object 112
in tick object 163
label extents
in percent widget 70
label function resource
in tick object 163
labels, draw, resource
in tick object 163
layout
in LEDIcon widget 49
leading one resource
in LEDNumerical widget 63
leading plus/minus sign resource
in LEDNumerical widget 63
leading zeros resource
in LEDNumerical widget 63
LED start value resource
in LEDIcon widget 53
LED value resource
in LEDIcon widget 53
LEDIcon widget 1, 47
LEDIcon widget resources
XmNbackground 51
XmNxrtGaBorder 51
XmNxrtGaDebug 51
XmNxrtGaDecayRate 51
XmNxrtGaDirection 51
XmNxrtGaDisplayMode 51
XmNxrtGaDoubleBuffer 51
XmNxrtGaExposeCallback 52
XmNxrtGaIconHeight 52
XmNxrtGaIconList 52
XmNxrtGaIconSpacing 52
XmNxrtGaIconWidth 52
XmNxrtGaImage 52
XmNxrtGaImageLayout 52
XmNxrtGaInteractionCallback 52
XmNxrtGaInteractionType 52
XmNxrtGaLEDStartValue 53
XmNxrtGaLEDValue 53
XmNxrtGaMaxIcon 53
XmNxrtGaOrientation 53
XmNxrtGaRepaint 53
XmNxrtGaResizeCallback 53
XmNxrtGaSizingPolicy 54
XmNxrtGaSpacingHorizontal 54
XmNxrtGaSpacingVertical 54
XmNxrtGaValueChangedCallback 54
XmNxrtGaXOrigin 54
XmNxrtGaYOrigin 54
LEDNumerical widget 1, 57
LEDNumerical widget resources
XmNbackground 62
XmNxrtGaBorder 62
XmNxrtGaColorMixPercentage 62
XmNxrtGaDebug 62
XmNxrtGaDecimalPlaces 62
XmNxrtGaDecimalPosition 62
XmNxrtGaDecimalSize 62
XmNxrtGaDoubleBuffer 62
XmNxrtGaExposeCallback 62
XmNxrtGaFixedDecimal 62
XmNxrtGaImage 63
XmNxrtGaImageLayout 63
XmNxrtGaLeadingOne 63

XmNxrtGaLeadingPlusMinus 63
XmNxrtGaLeadingZeros 63
XmNxrtGaNumDigits 63
XmNxrtGaOnColor 63
XmNxrtGaRepaint 63
XmNxrtGaResizeCallback 63
XmNxrtGaSegmentSpacing 63
XmNxrtGaSegmentWidth 63
XmNxrtGaSlopeOffset 64
XmNxrtGaSpacingHorizontal 64
XmNxrtGaSpacingVertical 64
XmNxrtGaText 64
XmNxrtGaValue 64
XmNxrtGaValueChangedCallback 64
XmNxrtGaValueType 64
left margin resource
in frame object 97
length resource
in needle object 127
line display resource
in state object 141
line inner radius resource
in state object 141
line inner radius use default resource
in state object 141
line outer radius resource
in state object 141
line outer radius use default resource
in state object 142
line pattern resource
in state object 142
line thickness resource
in state object 141, 142
linear dial widget 1, 35
linear dial widget resources
XmNxrtGaIndicator 43
XmNxrtGaLinearScale 43
XmNxrtGaOffColor 43
XmNxrtGaOffImage 44
XmNxrtGaOnColor 44
XmNxrtGaOnImage 44
XmNxrtGaShape 44
linear dial/gauge widget resources
XmNbackground 41
XmNxrtGaBorder 41
XmNxrtGaDebug 41
XmNxrtGaDoubleBuffer 41
XmNxrtGaExposeCallback 41
XmNxrtGaFrame 42
XmNxrtGaImage 42
XmNxrtGaImageLayout 42
XmNxrtGaInteractionCallback 42
XmNxrtGaMapCallback 42
XmNxrtGaRenderList 42
XmNxrtGaRepaint 42
XmNxrtGaResizeCallback 42
XmNxrtGaTextList 42
XmNxrtGaTickList 43
XmNxrtGaUseRenderList 43
linear gauge widget 1, 35
linear gauge widget resources
XmNxrtGaIndicatorList 43
XmNxrtGaLinearScaleList 43
XmNxrtGaRangeList 43
linear scale list resource
in linear gauge widget 43
linear scale object 115
linear scale object resources
XmNbackground 118
XmNheight 118
XmNwidth 118
XmNx 118
XmNxrtGaAllowTransform 118
XmNxrtGaDebug 118
XmNxrtGaDirection 118
XmNxrtGaMax 118
XmNxrtGaMin 118
XmNxrtGaOrientation 119
XmNxrtGaTransformCallback 119
XmNy 119
linear scale resource
in indicator object 106
in linear dial widget 43
linking
shared libraries 11

M

macros
COLOR_RES 217
DVAL 217
map callback resource
in circular dial/gauge widget 29
in linear dial/gauge widget 42
in percent widget 74
map function 174
Map() action 175
mapping 174
marker object 121
marker object resources
XmNbackground 123
XmNwidth 123
XmNxrtGaCircularScale 123
XmNxrtGaDebug 123
XmNxrtGaDrawFunc 123
XmNxrtGaImage 123
XmNxrtGaImageRotate 123
XmNxrtGaInnerRadius 123
XmNxrtGaIsShowing 123
XmNxrtGaMarkerType 123
XmNxrtGaOuterRadius 124
marker resource
in circular dial widget 31
marker type resource
in marker object 123
max icon resource
in LEDIcon widget 53
max resource
in circular scale object 84

in linear scale object 118
maximum value
attained in LEDIcon widget 48
methods
XmCreateXrtGaCircularDial 32
XmCreateXrtGaCircularGauge 32
XmIsXrtGaCircularDial 32
XmIsXrtGaCircularGauge 33
XrtGaArgValToDouble 217
XrtGaDoubleToArgVal 217
XrtGaDrawPolyTransform 218
XrtGaEnableImageType 218
XrtGaGetImage 218
XrtGaMap 218
XrtGaOutputXwd 219
XrtGaSaveImageAsJpeg 219
XrtGaSetXmStringConverter 220
XrtGaUnmap 220
min resource
in circular scale object 84
in linear scale object 118
minimum size, image, resource
in text object 156
mirror image
of an indicator shape 104
monitoring user interaction 171
multiple turn enabled resource
in switch object 149
multiple turn value resource
in switch object 149

N

needle list resource
in circular gauge widget 30
needle object 125
needle object resources
XmNbackground 127
XmNwidth 127
XmNxrtGaCircularScale 127
XmNxrtGaDebug 127
XmNxrtGaDrawFunc 127
XmNxrtGaInteractionType 127
XmNxrtGaIsShowing 127
XmNxrtGaLength 127
XmNxrtGaNeedleType 128
XmNxrtGaValue 128
XmNxrtGaValueChangedCallback 128
needle type resource
in needle object 128
num digits resource
in counter object 92
in LEDNumerical widget 63
numbering precision 161
numeric values
displaying as binary numbers in LEDIcon widget 48

O

objects
centering 15
in XRT/gauge 13
off color resource
in icon object 101
in linear dial widget 43
off image resource
in icon object 101
in linear dial widget 44
offset distance resource
in circular dial widget 31
offset resource
in range object 134
in tick object 164
offset type resource
in circular dial widget 31
offset use default resource
in state object 142
offset, x, resource
in state object 142
offset, y, resource
in state object 143
on color resource
in icon object 101
in LEDNumerical widget 63
in linear dial widget 44
on image resource
in icon object 101
in linear dial widget 44
Online Support Resources 3
orientation
of a linear scale object 116
orientation resource
in LEDIcon widget 53
in linear scale object 119
outer extent resource
in indicator object 106
in range object 134
in tick object 164
outer radius resource
in circular dial widget 31
in marker object 124
outer radius, line, resource
in state object 141

P

pattern, line, resource
in state object 142
percent mode 69
percent widget 1
percent widget resources
XmNbackground 73
XmNxrtGaBorder 73
XmNxrtGaCalcMode 73
XmNxrtGaDebug 73
XmNxrtGaDoubleBuffer 73

XmNxrtGaExposeCallback 73
XmNxrtGaFrame 73
XmNxrtGaImage 73
XmNxrtGaImageLayout 73
XmNxrtGaItemList 73
XmNxrtGaMapCallback 74
XmNxrtGaRenderList 74
XmNxrtGaRepaint 74
XmNxrtGaResizeCallback 74
XmNxrtGaScale 74
XmNxrtGaTextList 74
XmNxrtGaUseRenderList 74
precision 161
precision resource
in tick object 164
precision use default resource
in tick object 164
product feedback 3

R

radius resource
in center object 79
in switch object 150
radius size resource
in circular scale object 84
radius, inner, resource
in marker object 123
radius, outer, resource in marker object 124
range list resource
in circular dial/gauge widget 29
in linear gauge widget 43
range object 129
range object resources
XmNbackground 133
XmNxrtGaDebug 133
XmNxrtGaImage 134
XmNxrtGaImageLayout 134
XmNxrtGaInnerExtent 134
XmNxrtGaIsShowing 134
XmNxrtGaOffset 134
XmNxrtGaOuterExtent 134
XmNxrtGaScale 134
XmNxrtGaShape 134
XmNxrtGaStartValue 134
XmNxrtGaStopValue 135
ranges
adding to a linear gauge widget 38
in linear dials and gauges 37, 38
ratio mode 68
render list resource
in circular dial/gauge widget 29
in linear dial/gauge widget 42
in percent widget 74
rendering objects in linear dials and gauges 36
repaint resource
in circular dial/gauge widget 29
in LEDIcon widget 53
in LEDNumerical widget 63
in linear dial/gauge widget 42
in percent widget 74
reset alignment resource
in counter object 92
reset background color resource
in counter object 92
reset height resource
in counter object 92
reset image resource
in counter object 92
reset offset resource
in counter object 92
reset shape resource
in counter object 92
reset show resource
in counter object 92
reset width resource
in counter object 93
ResetCountert() action 173
Resize callback 179
resize callback resource
in circular dial/gauge widget 29
in LEDIcon widget 53
in LEDNumerical widget 63
in linear dial/gauge widget 42
in percent widget 74
resizing windows 179
resource files
comments in 247
continuing lines 247
resource syntax 247
sample program 249
specifying Booleans 247
specifying enumerated values 247
specifying images 248
specifying objects 248
specifying strings 248
use of 247
resource type/class listing 229
resource types and classes
center object 229
circular dial widget 229
circular gauge widget 230
circular scale object 231
counter object 232
frame object 233
icon object 233
indicator object 233
item object 234
LEDIcon widget 234
LEDNumerical widget 235
linear dial widget 236
linear gauge widget 237
linear scale object 238
marker object 238
needle object 239
percent widget 239
range object 240
state object 240
switch object 241

text object 242
tick object 243
resources
center object 79
circular dial widget 30
circular dial/gauge widget 28
circular gauge widget 30
circular scale object 84
counter object 90
frame object 96
icon object 100
indicator object 105
item object 111
LEDIcon widget 51
LEDNumerical widget 62
linear dial widget 43
linear dial/gauge widget 41
linear gauge widget 43
linear scale object 118
marker object 123
needle object 127
percent widget 72
range object 133
state object 140
switch object 148
text object 155
tick object 162
right margin resource
in frame object 97
rotate, image, resource
in indicator object 105
in marker object 123
in switch object 149
rotating
an image used in a switch 146
rotation direction
for a circular scale object 83

S

sample code 267
scale
in percent widget 69
scale resource
in percent widget 74
in range object 134
in tick object 164
scale, circular, resource
in needle object 127
in state object 141
in switch object 148
scale, circular, resource in marker object 123
scale, linear, resource
in indicator object 106
segment spacing resource
in LEDNumerical widget 63
segment width resource
in LEDNumerical widget 63
shape resource
in icon object 101
in linear dial widget 44
in range object 134
shared libraries
linking and authorizing 11
show reset resource
in counter object 92
sizing policy resource
in LEDIcon widget 54
slanted numerals
in LEDNumerical widget 60
slope offset resource
in LEDNumerical widget 64
spacing horizontal resource
in counter object 93
in LEDIcon widget 54
in LEDNumerical widget 64
spacing vertical resource
in counter object 93
in LEDIcon widget 54
in LEDNumerical widget 64
stages of user interaction 168
start angle resource
in circular scale object 84
start value resource
in range object 134
in tick object 164
state list resource
in circular dial widget 32
state object 137
state object resources
XmNfontList 140
XmNforeground 140
XmNxrtGaAlignment 140
XmNxrtGaAlignmentUseDefault 140
XmNxrtGaAngle 141
XmNxrtGaAngleUseDefault 141
XmNxrtGaCircularScale 141
XmNxrtGaDebug 141
XmNxrtGaIsShowing 139, 141
XmNxrtGaLineDisplay 141
XmNxrtGaLineInnerRadius 141
XmNxrtGaLineInnerRadiusUseDefault 141
XmNxrtGaLineOuterRadius 141
XmNxrtGaLineOuterRadiusUseDefault 142
XmNxrtGaLinePattern 142
XmNxrtGaLineThickness 141, 142
XmNxrtGaOffsetUseDefault 142
XmNxrtGaOffsetX 142
XmNxrtGaOffsetY 143
XmNxrtGaText 143
stop angle resource
in circular scale object 84
stop value resource
in range object 135
in tick object 165
string mode resource
in text object 157
string resource
in text object 157

structures 226
XrtGaCallbackStruct 226
XrtGaDrawData 226
XrtGaInteractionCallbackStruct 226
XrtGaMapCallbackStruct 226
XrtGaMapResult 227
XrtGaPolyDrawStruct 227
XrtGaTransformCallbackStruct 227
XrtGaUserDrawInfo 228
XrtGaValueChangedCallbackStruct 228
switch object 145
switch object resources
XmNbackground 148
XmNxrtGaCircularScale 148
XmNxrtGaContinuous 148
XmNxrtGaDebug 148
XmNxrtGaDrawFunc 148
XmNxrtGaImage 149
XmNxrtGaImageRotate 149
XmNxrtGaIncrementValue 149
XmNxrtGaInteractionType 149
XmNxrtGaIsShowing 149
XmNxrtGaMultipleTurnEnabled 149
XmNxrtGaMultipleTurnValue 149
XmNxrtGaRadius 150
XmNxrtGaSwitchType 150
XmNxrtGaValue 150
XmNxrtGaValueChangedCallback 150
switch resource
in circular dial widget 32
switch type resource
in switch object 150

T

technical support 2
text list resource
in circular dial/gauge widget 29
in linear dial/gauge widget 42
in percent widget 74
text object 153
text object resources
XmNbackground 155
XmNfontList 155
XmNforeground 155
XmNheight 155
XmNwidth 155
XmNx 155
XmNxrtGaAlignment 155
XmNxrtGaAllowWidgetPositioning 155
XmNxrtGaBorder 156
XmNxrtGaBorderWidth 156
XmNxrtGaDebug 156
XmNxrtGaImage 156
XmNxrtGaImageLayout 156
XmNxrtGaImageMinimumSize 156
XmNxrtGaIsShowing 156
XmNxrtGaString 157
XmNxrtGaStringMode 157
XmNxrtGaValueObject 157
XmNy 157
text resource
in LEDNumerical widget 64
in state object 143
text, adding 154
thickness, line, resource
in state object 141, 142
tick list resource
in circular dial/gauge widget 30
in linear dial/gauge widget 43
tick object 159
tick object resources
XmNbackground 162
XmNfontList 162
XmNwidth 162
XmNxrtGaAutomatic 162
XmNxrtGaDebug 163
XmNxrtGaDrawLabels 163
XmNxrtGaIncrementValue 163
XmNxrtGaInnerExtent 163
XmNxrtGaIsShowing 163
XmNxrtGaLabelExtent 163
XmNxrtGaLabelFunc 163
XmNxrtGaOffset 164
XmNxrtGaOuterExtent 164
XmNxrtGaPrecision 164
XmNxrtGaPrecisionUseDefault 164
XmNxrtGaScale 164
XmNxrtGaStartValue 164
XmNxrtGaStopValue 165
XmNxrtGaTickType 165
tick type resource
in tick object 165
top margin resource
in frame object 98
transform callback 171, 172
transform callback resource
in circular scale object 85
in linear scale object 119
transition rate resource
in counter object 93
transition steps resource
in counter object 93
TranslateStart() action 170
TranslateUpdate() action 169, 172
translation, interactive 167
translations and actions 167
disabling 169, 171
monitoring 171
programming overview 167
translation 167
zoom 167
see also user interaction

U

UIL arguments
center object resources 251

circular dial widget resources 252
circular gauge widget resources 252
circular scale object resources 253
counter object resources 254
frame object resources 255
icon object resources 255
indicator object resources 255
item object resources 256
LEDIcon widget resources 256
LEDNumerical widget resources 258
linear dial widget resources 259
linear gauge widget resources 259
linear scale object resources 260
marker object resources 260
needle object resources 261
percent widget resources 261
range object resources 262
state object resources 262
switch object resources 263
text object resources 263
tick object resources 264
use default angle resource
in state object 141
use default precision resource
in tick object 164
use default resource
in state object 140
use default, line innner radius, resource
in state object 141
use default, line outer radius, resource
in state object 142
use default, offset, resource
in state object 142
use render list resource
in circular dial/gauge widget 30
in linear dial/gauge widget 43
in percent widget 74
use transition resource
in counter object 93
user interaction
controlling translate, zoom 170, 172
customizing 167
disabling 171
disabling all 169
effect on XRT/Gauge resources 172
event handling 178
starting 168
translating 170, 172
translation 167
updating 169
zooming 167, 170, 172
user interaction in XRT/gauge 167
user interaction, programming interactions
three stages 168
using GUI Builders 247
using resource files 12, 247

V

value
of LEDNumerical widget 58
value changed callback 178
value changed callback resource
in counter object 93
in indicator object 106
in item object 112
in LEDNumerical widget 64
in needle object 128
in switch object 150
value changed resource
in LEDIcon widget 54
value object resource
in text object 157
value resource
in counter object 93
in indicator object 106
in item object 112
in LEDNumerical widget 64
in needle object 128
in switch object 150
value type resource
in LEDNumerical widget 64
vertical spacing resource
in counter object 93
in LEDIcon widget 54
in LEDNumerical widget 64

W

widget
introduction 1
Motif class hierarchy 10
widget positioning, allow, resource
in text object 155
widget synopses
circular dial widget 27
circular gauge widget 27
LEDIcon widget 50
LEDNumerical widget 61
linear dial widget 41
linear gauge widget 40
widgets, in XRT/gauge
class and constructor names 12
description 7
width resource
in counter object 90
in indicator object 105
in linear scale object 118
in marker object 123
in needle object 127
in text object 155
in tick object 162
width, border, resource in text object 156
windows, resizing 179

X

x coordinate resource
  in counter object 90
  in linear scale object 118
  in text object 155
x offset resource
  in state object 142
x origin resource
  in circular scale object 85
  in LEDIcon widget 54
XmCreateXrtGaCenter 79
XmCreateXrtGaCircularDial 32
XmCreateXrtGaCircularGauge 32
XmCreateXrtGaCounter() 93
XmCreateXrtGaFrame 98
XmCreateXrtGaIcon 101
XmCreateXrtGaIndicator 107
XmCreateXrtGaItem 112
XmCreateXrtGaLinearScale 119
XmCreateXrtGaMarker 124
XmCreateXrtGaNeedle 128
XmCreateXrtGaPercent 75
XmCreateXrtGaRange 135
XmCreateXrtGaState 143
XmCreateXrtGaSwitch 150
XmCreateXrtGaTick 165
XmCreateXrtTable() 217
XmIsXrtGaCenter 80
XmIsXrtGaCircularDial 32
XmIsXrtGaCircularGauge 33
XmIsXrtGaCounter 93
XmIsXrtGaFrame 98
XmIsXrtGaIcon 102
XmIsXrtGaIndicator 107
XmIsXrtGaItem 113
XmIsXrtGaLinearScale 119
XmIsXrtGaMarker 124
XmIsXrtGaNeedle 128
XmIsXrtGaRange 135
XmIsXrtGaState 143
XmIsXrtGaSwitch 151
XmIsXrtGaTick 165
XmNbackground 28, 41, 51, 62, 73, 79, 84, 90, 97, 100, 105, 111, 118, 123, 127, 133, 148, 155, 162
XmNfontList 90, 140, 155, 162
XmNforeground 140, 155
XmNheight 90, 118, 155
  in LEDNumerical widget 58
XmNwidth 90, 105, 118, 123, 127, 155, 162
  in LEDNumerical widget 58
  resource in the indicator object 104
XmNx 90, 118, 155
XmNxrtGaAlignment 140, 155
XmNxrtGaAlignmentUseDefault 140
XmNxrtGaAllowTransform 84, 118
XmNxrtGaAllowWidgetPositioning 155
XmNxrtGaAngle 141
XmNxrtGaAngleUseDefault 141
XmNxrtGaAutomatic 162
XmNxrtGaBorder 28, 41, 51, 62, 73, 91, 156
XmNxrtGaBorderWidth 91, 156
XmNxrtGaBottomMargin 97
XmNxrtGaCalcMode 73
XmNxrtGaCenterList 30
XmNxrtGaCircularScale 30, 79, 123, 127, 141, 148
XmNxrtGaCircularScaleList 30
XmNxrtGaColorMixPercentage 62
  in LEDNumerical widget 58
XmNxrtGaContinuous 148
XmNxrtGaCounterList 28
XmNxrtGaDebug 28, 41, 51, 62, 73, 79, 84, 91, 97, 101, 105, 111, 118, 123, 127, 133, 141, 148, 156, 163
XmNxrtGaDecayRate 51
XmNxrtGaDecimalPlaces 62, 91
XmNxrtGaDecimalPosition 62
XmNxrtGaDecimalSize 62
XmNxrtGaDigitBackgroundColor 91
XmNxrtGaDigitForegroundColor 91
XmNxrtGaDirection 51, 83, 84, 118
XmNxrtGaDisplayMode 51
XmNxrtGaDoubleBuffer 28, 41, 51, 62, 73
XmNxrtGaDrawFunc 105, 123, 127, 148
XmNxrtGaDrawLabels 163
XmNxrtGaExposeCallback 28, 41, 52, 62, 73, 179
  in linear dials and gauges 37
XmNxrtGaFixedDecimal 62
XmNxrtGaFrame 28, 42, 73
XmNxrtGaHighlightColor 30
XmNxrtGaHighlightSelectedState 30
XmNxrtGaIconHeight 52
  in LEDIcon widget 49
XmNxrtGaIconList 52
XmNxrtGaIconSpacing 52
XmNxrtGaIconWidth 52
  in LEDIcon widget 49
XmNxrtGaImage 29, 42, 52, 63, 73, 79, 91, 97, 105, 112, 123, 134, 149, 156
  in thepercent widget 72
XmNxrtGaImageLayout 29, 42, 52, 63, 73, 92, 97, 134, 156
  in thepercent widget 72
XmNxrtGaImageMinimumSize 156
XmNxrtGaImageRotate 105, 123, 146, 149
XmNxrtGaIncrementValue 149, 163
XmNxrtGaIndicator 43
XmNxrtGaIndicatorList 43
XmNxrtGaIndicatorType 106
XmNxrtGaInnerExtent 104, 106, 134, 163
XmNxrtGaInnerRadius 123
XmNxrtGaInteractionCallback 29, 42, 52, 169, 170
XmNxrtGaInteractionType 52, 106, 127, 149
XmNxrtGaIsShowing 79, 92, 97, 106, 112, 123, 127, 134, 139, 141, 149, 156, 163
XmNxrtGaItemList 70, 73
XmNxrtGaLabelExtent 112, 163
XmNxrtGaLabelFunc 163
XmNxrtGaLeadingOne 63

XmNxrtGaLeadingPlusMinus 63
XmNxrtGaLeadingZeros 63
XmNxrtGaLEDStartValue 53
XmNxrtGaLEDValue 53
XmNxrtGaLeftMargin 97
XmNxrtGaLength 127
XmNxrtGaLinearScale 43, 106
XmNxrtGaLinearScaleList 43
  in linear dials and gauges 37
XmNxrtGaLineDisplay 141
XmNxrtGaLineInnerRadius 31, 141
XmNxrtGaLineInnerRadiusUseDefault 141
XmNxrtGaLineOuterRadius 31, 141
XmNxrtGaLineOuterRadiusUseDefault 142
XmNxrtGaLinePattern 142
XmNxrtGaLineThickness 141, 142
XmNxrtGaMapCallback 29, 42, 74
XmNxrtGaMarker 31
XmNxrtGaMarkerType 123
XmNxrtGaMax 84, 118
XmNxrtGaMaxIcon 48, 53
XmNxrtGaMin 84, 118
XmNxrtGaMultipleTurnEnabled 149
XmNxrtGaMultipleTurnValue 149
XmNxrtGaNeedleList 30
XmNxrtGaNeedleType 128
XmNxrtGaNumDigits 63, 92
  in LEDNumerical widget 58
XmNxrtGaOffColor 43, 101
  in LEDNumerical widget 58
XmNxrtGaOffImage 44, 100, 101
  in linear dials and gauges 38
XmNxrtGaOffset 134, 164
XmNxrtGaOffsetDistance 31
XmNxrtGaOffsetType 31
XmNxrtGaOffsetUseDefault 142
XmNxrtGaOffsetX 142
XmNxrtGaOffsetY 143
XmNxrtGaOnColor 44, 63, 101
  in LEDNumerical widget 58
XmNxrtGaOnImage 44, 100, 101
XmNxrtGaOrientation 53, 119
XmNxrtGaOuterExtent 104, 106, 134, 164
XmNxrtGaOuterRadius 124
XmNxrtGaPrecision 164
XmNxrtGaPrecisionUseDefaul 164
XmNxrtGaRadius 79, 150
XmNxrtGaRadiusSize 84
XmNxrtGaRangeList 29, 43
XmNxrtGaRenderList 29, 42, 74
  in thepercent widget 72
XmNxrtGaRepaint 14, 29, 42, 53, 63, 74
  in thepercent widget 72
XmNxrtGaResetAlignment 92
XmNxrtGaResetBackgroundColor 92
XmNxrtGaResetHeight 92
XmNxrtGaResetImage 92
XmNxrtGaResetOffset 92
XmNxrtGaResetShape 92
XmNxrtGaResetShow 92
XmNxrtGaResetWidth 93
XmNxrtGaResizeCallback 29, 42, 53, 63, 74, 179
  in linear dials and gauges 37
XmNxrtGaRightMargin 97
XmNxrtGaScale 69, 74, 134, 164
XmNxrtGaSegmentSpacing 63
XmNxrtGaSegmentWidth 63
  in LEDNumerical widget 58
XmNxrtGaShape 44, 101, 134
XmNxrtGaSizingPolicy 54
XmNxrtGaSlopeOffset 64
XmNxrtGaSpacingHorizontal 54, 64, 93
  in LEDNumerical widget 59
XmNxrtGaSpacingVertical 54, 64, 93
XmNxrtGaStartAngle 84
XmNxrtGaStartValue 134, 164
XmNxrtGaStateList 32
XmNxrtGaStopAngle 84
XmNxrtGaStopValue 135, 165
XmNxrtGaString 157
XmNxrtGaStringMode 157
XmNxrtGaSwitch 32
XmNxrtGaSwitchType 150
XmNxrtGaText 64, 143
XmNxrtGaTextList 29, 42, 72, 74
  in linear dials and gauges 37
XmNxrtGaTickList 30, 43
  in linear dials and gauges 37
XmNxrtGaTickType 165
XmNxrtGaTopMargin 98
XmNxrtGaTransformCallback 85, 119, 171, 172
XmNxrtGaTransitionRate 93
XmNxrtGaTransitionSteps 93
XmNxrtGaUseRenderList 30, 36, 43, 74
  in thepercent widget 72
XmNxrtGaUseTransition 93
XmNxrtGaValue 64, 93, 106, 112, 128, 150
XmNxrtGaValueChangedCallback 54, 64, 93, 106, 112, 128, 150
XmNxrtGaValueObject 157
XmNxrtGaValueType 64
XmNxrtGaXOrigin 54, 85
  in LEDIcon widget 49
XmNxrtGaYOrigin 54, 85
  in LEDIcon widget 49
XmNxrtPDSSorted 207
XmNy 91, 119, 157
XRT/gauge
  and C++ 12
  and UIL 251
  circular dial widget 17
  circular gauge widget 17
  data types 221
  internationalization 13
  introduction 1
  linear dial widget 35
  linear gauge widget 35
  Motif class hierarchy 10
  resource types/classes 229
xrt_auth utility

flags 11
use of 11
XRTGA_ALIGNMENT_BEGINNING 138
XRTGA_ALIGNMENT_CENTER 138
XRTGA_ALIGNMENT_END 138
XrtGaAlignment enumeration 221
XrtGaBorder enumeration 221
XrtGaCalcMode enumeration 221
XrtGaCallbackStruct structure 226
XrtGaDirection enumeration 222
XrtGaDisplayMode enumeration 222
XrtGaDrawData structure 226
XrtGaFillPattern enumeration 222
XrtGaGetImage 218
XrtGaImageLayout enumeration 222
XrtGaIndicatorType enumeration 222
XrtGaInteractionCallbackStruct structure 226
XrtGaInteractionctionCallbackStruct structure 169, 170
XrtGaInteractionType enumeration 223
XrtGaLinePattern enumeration 223
XrtGaMap 218
XrtGaMapCallbackStruct structure 226
XrtGaMapResult structure 174, 227
XrtGaMarkerType enumeration 223
XrtGaNeedleType enumeration 223
XrtGaOffsetType enumeration 223
XrtGaOutputXwd 219
XrtGaPolyDrawStruct structure 227
XrtGaReason 224
XrtGaSaveImageAsJpeg 219
XrtGaSetXmStringConverter 220
XrtGaShape enumeration 224
XrtGaSizingPolicy enumeration 224
XrtGaStringMode enumeration 225
XrtGaSwitch 145
XrtGaSwitchType enumeration 225
XrtGaTickType enumeration 225
XrtGaTransformCallbackStruct structure 227
XrtGaUnmap 220
XrtGaUnmap() 175
XrtGaUserDrawInfo structure 228
XrtGaValueChangedCallbackStruct structure 228
XrtGaValueType enumeration 225
XrtPDSCmpDoubles() 199, 204
XrtPDSCmpFloats() 199, 204
XrtPDSCmpInts() 199, 204
XrtPDSCmpStrings() 199, 204
XrtPDSCmpStringsIgnoreCase() 200, 205
XrtPDSList
adding items 194
adding sorted items 195
checking for existence 197
comparing items 199
copying 194
creating 194
destroy procedure 196
destroying 194
destroying item memory 195
introduction 194
item count 199
moving items 200
removing items 195
retrieving entire array 198
retrieving item by index 198
sample code 194
searching 196
sorting 200
XrtPDSListAdd() 195, 205
XrtPDSListAppend() 195, 205
XrtPDSListCreate() 194, 195, 205
XrtPDSListDelete() 195, 206, 211
XrtPDSListDeleteAll() 206, 211
XrtPDSListDestroy() 194, 196, 206
XrtPDSListDestroyAll() 196
XrtPDSListDestroyObject() 196, 206
XrtPDSListDestroyXtPointer() 196, 207
XrtPDSListDuplicate() 194, 207
XrtPDSListFind() 197, 199, 207
XrtPDSListGetCompareProc() 197, 205, 207, 210, 211
XrtPDSListGetItem() 198, 208
XrtPDSListGetItemCount() 199, 208
XrtPDSListGetItems() 198, 208
XrtPDSListGetItemSize() 199, 205, 208, 209
XrtPDSListGetUserData() 201, 208
XrtPDSListInsertAfter() 195, 209
XrtPDSListInsertBefore(195
XrtPDSListInsertBefore() 195, 209
XrtPDSListIsList() 197, 209
XrtPDSListIsSorted() 200, 209
XrtPDSListLookupItem() 197, 210
XrtPDSListMoveItemsAfter() 210
XrtPDSListMoveItemsBefore() 200, 210
XrtPDSListRemove() 195, 206, 210
XrtPDSListRemoveAll() 195, 206, 211
XrtPDSListSearchBackward() 198
XrtPDSListSearchForward() 198, 211
XrtPDSListSearchReverse() 211
XrtPDSListSetCompareProc() 197, 199, 200, 204, 205, 212
XrtPDSListSetDestroyProc() 196, 207, 212
XrtPDSListSetItem() 198, 212
XrtPDSListSetUserData() 201, 212, 213
XrtPDSListSort() 200, 213
XrtPDSSetDestroyProc() 206, 211

Y

y coordinate resource
in counter object 91
in linear scale object 119
in text object 157
y offset resource
in state object 143
y origin resource
in circular scale object 85
in LEDIcon widget 54

Z

zeros, leading, resource
in LEDNumerical widget 63
ZoomEnd() action 172
zooming, interactive 167
ZoomStart() action 170, 172