XRT/graph™ for Motif

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 4.0



Software Development Productivity

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

January 1999

Part No. KLGRM40

Copyright © 1991-1999 by KL Group Inc. All Rights Reserved.

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

Printed in Canada on recycled paper. 

A Note on XRT Product Licensing

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

- XRT products (such as XRT/3d, XRT/graph) are licensed on a per toolkit (such as Motif) and per architecture (such as SPARC, RS6000, HP 700/800, OpenVMS AXP, etc.) basis.
- There are two types of XRT licenses: single and network-site.
- A single license authorizes use of an XRT product for development of end-user applications on a particular machine identified by a CPU ID. Development activities can include the following:
  - compiling and linking the XRT product library into an end-user application binary.
  - running the XRT Builder tool (if one is supplied).
  - integrating the XRT product into a GUI Builder or UIMS tool. Once an XRT product is integrated into such a tool, you may use the tool for XRT development only on machines that are licensed to use the XRT product.
- A network-site development license authorizes use of an XRT product for development of end-user applications on all similar-architecture machines on the same network and located in the same building.
- You may distribute end-user application binaries including the statically-linked XRT product library anywhere, and as many times as you like, without incurring royalty or run-time fees.

XRT FOR MOTIF BINARY LICENSE AGREEMENT

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

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

SOFTWARE LICENSE

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

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

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

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

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

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

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

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

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

LIMITED WARRANTY

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

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

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

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

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

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

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

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

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

Table of Contents

Preface 1
Introduction 1
New Features in XRT/graph 4.0 2
Assumptions 3
Typographical Conventions Used in This Manual 3
Related Documents 3
Technical Support 4
Product Feedback and Announcements 4

Part I: Using XRT/graph

1 Getting Started: Developing a Simple XRT/graph Program 9
1.1 Introduction 9
1.2 Environment Variables 9
1.3 Compiling and Authorizing a Program 10
1.4 A Basic Plot 11
1.5 Loading Data From a File 13
1.6 Adding Header, Footer, and Labels 13
1.7 Changing to a Bar Chart 15
1.8 Inverting Graph Orientation 16
1.9 Putting it all together 17
1.10 Sample Code 18

2 XRT/graph Basics 21
2.1 Integration with Motif 21
2.2 Terminology 21
2.3 Graph Types 24
2.4 Floating Point Resources 27
2.5 UseDefault Resources 27
2.6 Pointer Resources 28
2.7 String Resources 28
2.8 Colors 29

2.9 Using Resource Files 31

3 Using the Property Editor 33

3.1 Getting Started 34
3.2 Using the Property Editor 35
3.3 Programming the Property Editor 36
Customizing End-User Property Editor 36
Popup, Popdown, and Update Programatically 37
Loading and Saving Graph Descriptions 37
Property Editor Source Code 38
3.4 Generating Source Code 38
Files Created by Source Code Generator 39
Compiling and Running Generated Code 41

4 Programming XRT/graph 43

4.1 Resource Summary 43
4.2 Axis Labelling and Annotation Methods 50
Values Annotation 50
Point-labels Annotation and Set-labels 52
Value-labels Annotation 54
Time-axis Labels Annotation 56
4.3 Axis Controls 60
Axis and Data Bounds 60
Origin 61
Tick and Grid Increments 62
Axis Title and Annotation Rotation 63
Other Axis Controls 63
4.4 Annotation Formatting 66
4.5 Positioning Graph Areas 69
4.6 3D Effect 70
4.7 Stacking 72
4.8 Header, Footer and Legend Display 72
4.9 Data Styles 74
4.10 Special Bar Chart Resources 80
4.11 Special Pie Chart Resources 82
4.12 HiLo, Candle and Bubble Graphs 84
4.13 Special Polar/Radar Graph Resources 87
4.14 Foreground and Background Colors 88
4.15 Markers 89
4.16 Area Borders 91
4.17 Background Images 91

4.18 Double Buffering 93
4.19 Adding Scrollbars to a Graph 94
4.20 Output and Printing 95

5 XRT/graph Data 97
5.1 Getting Data into Graphs 97
5.2 The Data Object 98
Array and General Data Formats 99
Basic XrtDataHandle Attributes 100
5.3 Object and Data Access Methods 100
5.4 Excluding and Hiding Data 103
5.5 Transposing the Data 104
5.6 Using Multiple Datasets 105
5.7 Example Using Static Data 106
5.8 Example Reading Real-time Data from a Pipe 107

6 Programming User Interaction 109
6.1 Default User Interaction 109
6.2 Overview of Translations and Actions 109
6.3 Starting User Interaction 111
6.4 Updating User Interaction 112
Scaling, Translation, and Graphics Zooming 112
Axis Zooming 113
Resetting Scaling, Translation, and Zooming 114
Rotation 115
Disabling and Disallowing Interactions 116
6.5 Ending User Interaction 116
6.6 Interacting with Graph Data 117
Action Routines and Callbacks 117
Event Handling Procedures 119
6.7 Window Resizing 125

7 Advanced Programming Topics 127
7.1 Adding a Second Y-axis 127
7.2 Combination Graphs 128
7.3 Adding Text Objects 130
7.4 Batching Resource Updates 133
7.5 Internationalization Support 133
7.6 Fast Update Procedures 135
7.7 Porting XRT/graph to Microsoft Windows 136
7.8 Customizing Warning Messages 140

7.9 Using C++ with XRT/graph 140
7.10 Using UIL with XRT/graph 141
7.11 Converting Resource Values to Strings 141

8 PDS Utility Library 143
8.1 XrtPDSImage 143
Creating an XrtPDSImage 143
8.2 Registering Image Types 144
Predefined Image Support 144
Custom Image Support 145
Writing the Image Functions 147
8.3 Using Dynamic or Shared Libraries 148
8.4 XrtPDSList Object 148
Sample Code 149
Creating and Destroying a List 149
Adding and Deleting List Items 149
Searching and Retrieving List Items 151
List Comparison Routines 154
Sorting a List 155
User-Defined Data 156
8.5 Procedures and Methods Reference 156
XrtPDSImage Methods 156
XrtPDSList Object Methods 158

Part II: Reference Appendices

A Resource Reference 171
A.1 Widget Synopsis 171
A.2 XRT/graph Resources 171
A.3 Inherited Resources 203

B Text Object Attribute Reference 205

C PointStyle Object Attribute Reference 209

D Procedures and Methods Reference 211

E Translations and Actions Reference 251
E.1 Default Translations 251
E.2 Action Routines 252

F Data Types 257

G Resource Names, Types and Classes 267

H Resource File and GUI Builder Usage 277

H.1 Syntax 277
H.2 Example Resource File 280

I UIL Arguments 283

Index 289

Preface

Introduction ■ *New Features in XRT/graph 4.0*
Assumptions ■ *Typographical Conventions Used in This Manual*
Related Documents ■ *Technical Support*
Product Feedback and Announcements

Introduction

Motif is composed of many different widgets such as buttons, sliders, and menus. Conceptually, XRT/graph adds a new widget to Motif. The graph widget displays data graphically in a window and can interact with a user.

The graph widget has resources that determine how the graph will look and behave. Writing programs using XRT/graph is similar to writing any other kind of Motif program; but now you have one more Motif widget to work with.

XRT/graph may be used in conjunction with KL Group's XRT/3d and XRT/table widgets. XRT/3d displays 3D data in surface, contour, scatter and bar graph representations. XRT/table displays rows and columns of tabular information in a scrollable window.

XRT/graph has resources which allow control of:

- Graph type (plot, area, bar, stacking-bar, pie, bubble, Hilo, HiLoOpenClose, candle, polar, radar, filled-radar, or combination).
- Header and footer positioning, border style, text, font, color and background image.
- Data styles: line colors and patterns, fill colors and patterns, line thickness, individual point style, size, and colors.
- Legend positioning, orientation, border style, anchor, font, color, and title.
- Graph positioning, border style, color, width, height, and 3D effect.
- Axis labeling using Point-labels, Set-labels, Value-labels, or Time-labels.
- Axis and data minimums and maximums, axis numbering methods, numbering and ticking increments, grid increments, font, origins, axis directions and precisions.
- Placement of axes, annotation, and origins. Inversion of axes.

- Management of user-interaction with the widget using callback resources.
- Markers and Text Areas.

XRT/graph also provides several procedures and methods which:

- Allocate and load data objects containing the numbers to be displayed.
- Update elements and data contained in data objects.
- Allow fast display of new data for specific chart types.
- Provide facilities to output graphs as JPEG or PNG image files.
- Enable a property editor that developers or end-users can use to view and change graph resources at runtime.
- Output a representation of the graph in PostScript format.
- Assist the developer with setting and getting indexed resources.

New Features in XRT/graph 4.0

New/Enhanced Graph Types

- Bubble charts
- Stacking and 100% added to area, plot, radar and filled radar types
- Exploding/missing pie slices
- User-specified pie starting angle

Visual Enhancements

- Data styles can now be changed on a *per point* basis.
- Axis datastyles are now provided, one per axis.
- Legends can now have a title.
- There are now more point styles and default datastyles.
- Charts, headers, footers and text areas can now have background images.

PDS Utility Library

Support has been added for output to JPEG, PNG or any other user-defined image type via the new PDS Utility Library. XrtPDSImage is an opaque generic object which contains an image. Pixmaps are still used internally, but this revolutionary approach allows unlimited image type support, since the user defines (registers) the type of image they want support for and XRT/graph only has to handle the generic object.

The Library also includes support for XrtPDSList objects. Taking a page from XRT/gear, this feature provides a convenient way to store groups of items.

Other Enhancements

- New and improved demos and examples.

- Bug fixes and minor improvements (see the *Release and Installation Notes* document for a listing of the main bugs fixed).

Assumptions

This manual assumes that the reader is 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/graph applications. An understanding of basic Motif programming concepts such as getting and setting resource values is required before continuing with this manual. See "Related Documents" below for information on Motif programming references.

Typographical Conventions Used in This Manual

`Typewriter Font`	■ C language source code and examples of file contents. ■ XRT/graph and Motif resource names and constants. ■ Commands that you enter at a terminal emulation 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*.
`...[XYY2]...`	Many XRT/graph resources used to specify axis information are similar for the X-, Y-, and Y2-axis. The syntax `[XYY2]` used in a resource name refers to all of the X, Y, or Y2 versions of this resource. For example, "`XtNxrtXNum`, `XtNxrtYNum`, and `XtNxrtY2Num`" is shortened to "`XtNxrt[XYY2]Num`".

Related Documents

Motif Programming Manual from O'Reilly & Associates, Inc.

Xlib Programming Manual and *Xlib Reference Manual* from O'Reilly & Associates, Inc. These manuals are not required to develop applications using XRT/graph 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/graph. You may purchase annual XRT/care agreements from your XRT Reseller or KL Group. XRT/care ensures you will receive all XRT/graph 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/graph from KL Group, contact KL Group for technical support; if you purchased XRT/graph 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/graph license serial number. Collecting the following additional information will help us serve you better:

- Identification – your name, company name, case number (if in regards to a previous case)
- Product – XRT/graph 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/graph, any problems you encounter, or any additional features you would find helpful. The majority of enhancements to XRT products are the result of customer requests.

Please send your comments to:

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

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

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

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

Part I

Using XRT/graph

1

Getting Started: Developing a Simple XRT/graph Program

Introduction ■ *Environment Variables*
Compiling and Authorizing a Program ■ *A Basic Plot*
Loading Data From a File ■ *Adding Header, Footer, and Labels*
Changing to a Bar Chart ■ *Inverting Graph Orientation*
Putting it all together ■ *Sample Code*

1.1 Introduction

This chapter allows you to immediately try out XRT/graph by compiling and running an example program. This program, *plot1.c*, will graph the 1963 Quarterly Expenses and Revenues for "Michelle's Microchips", a small company a little ahead of its time.

The following table shows the data to be displayed:

	Q1	Q2	Q3	Q4
Expenses	150.0	175.0	160.0	170.0
Revenue	125.0	100.0	225.0	300.0

1.2 Environment Variables

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

Before attempting to use XRT/graph, 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/graph.

1.3 Compiling and Authorizing a Program

If the X, Motif and XRT/graph libraries and include files are installed in the standard places (normally */usr/lib*, */usr/include/X11*, and */usr/include/Xm)*, *plot1.c* may be compiled and authorized by entering:[1]

```
cc plot1.c -I$XRTHOME/include \
    -L$XRTHOME/lib -o plot1 \
    -lxrtm -lpdsutil -lXm -lXpm -lXt -lX11 -lm
$XRTHOME/bin/xrt_auth plot1
```

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

xrt_auth

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

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

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

If your application is not successfully enabled using *xrt_auth*, graph widgets will display for a few seconds followed by the message shown in Figure 1. A more detailed message with troubleshooting tips is also written to *stdout*.

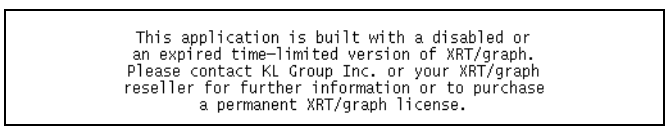


Figure 1 Message Displayed by an Expired or Non-authorized Application

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

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

C++ and ANSI C

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

1.4 A Basic Plot

Figure 2 lists the program *plot1.c*. This program is a minimal C program which loads the data from a file, creates a Motif shell widget and then creates a graph widget that will plot the data. As with many of the example programs in this manual, this file is located in the *$XRTHOME/src/graph/examples* directory.

```
1 #include <X11/Intrinsic.h>
2 #include <Xm/Xm.h>
3 #include <Xm/XrtGraph.h>
4
5 int
6 main(int argc, char *argv[])
7
8 {
9   XtAppContext    app_context;
10  Widget          graph, toplevel;
11  XrtDataHandle   hData;
12
13  /* Allocate and load the data */
14  hData = XrtDataCreateFromFile("data/mm63.dat", NULL);
15
16  /* Initialize Intrinsics */
17  toplevel = XtVaAppInitialize(&app_context, "Plot 1", NULL, 0,
18          (Cardinal *) &argc, argv, NULL, NULL);
19
20  /* Create graph */
21  graph = XtVaCreateManagedWidget("mygraph",
22          xtXrtGraphWidgetClass,  toplevel,
23          XtNxrtData,             hData,
24          NULL);
25
26  XtRealizeWidget(toplevel);
27  XtAppMainLoop(app_context);
28 }
```

Figure 2 plot1.c

Most of the code in *plot1.c* should be familiar to Motif programmers. The statement on line 14 allocates a data object (referenced by the `XrtDataHandle hData`) and loads that object with the data from the file *data/mm63.dat*. Make sure there is a *data* directory below the current directory when this program is executed.

On line 17, the Intrinsics are initialized and a toplevel shell widget is created in the standard Xt Intrinsics fashion. On lines 21-24, the Xt Intrinsics procedure *XtVaCreateManagedWidget()*[1] is used to instantiate a graph widget which is a managed child of `toplevel`. When the graph is created, the `XtNxrtData` resource is set to `hData`.

The standard Xt procedures (like *XtSetValues()*, *XtGetValues()*, *XtVaSetValues()*) may be used to manipulate the graph widget. The graph widget is subclassed from the Motif Primitive class, so in addition to having graph resources, it also has Primitive and Core resources. For example, it is possible to set the `XmNshadowThickness` resource of a graph widget.

XRT/graph supports 12 different *types* of graphs: plots, area graphs, bar, bubble, stacking bar, pie, HiLo, HiLoOpenClose, candle, polar, radar, and filled radar charts. The type is specified with the `XtNxrtType` resource. Because plot is the default, we do not have to set the `XtNxrtType` resource in *plot1.c*.

When the program is compiled and run, the window shown in Figure 3 displays.

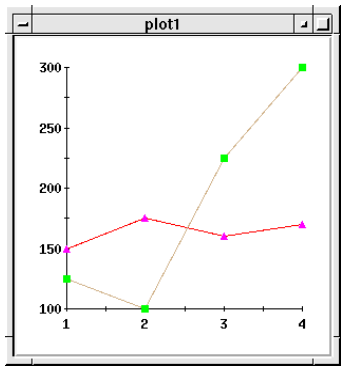


Figure 3 The plot1 Window

1. In this manual, the Varargs interface is used whenever possible. There is a Varargs interface for widget creation (*XtVaCreateManagedWidget()*, *XtVaCreateWidget()*) and setting and getting resources (*XtVaSetValues()*, *XtVaGetValues()*). See your Xt Intrinsics documentation for more information on the Varargs interface.

1.5 Loading Data From a File

A common task in any XRT/graph program is to load the graph data (called a *dataset*) into a format that the graph can use. To allocate and load the data from a file, use the *XrtDataCreateFromFile()* procedure.[1]

Figure 4 shows the contents of the *mm63.dat* file. This file is in the format understood by **XrtDataCreateFromFile**(). Lines beginning with a '#' symbol are treated as comments. The first line tells **XrtDataCreateFromFile**() that the data which follows is in `XRT_DATA_ARRAY` layout, and is made up of 2 *sets* of 4 *points*. XRT/graph supports two types of data organization: `XRT_DATA_ARRAY` and `XRT_DATA_GENERAL`.

```
ARRAY 2 4
# X-values
1.0 2.0 3.0 4.0
# Y-values set 1
150.0 175.0 160.0 170.0
# Y-values set 2
125.0 100.0 225.0 300.0
```

Figure 4 The mm63.dat file

The *mm63.dat* file defines 2 *sets* of 4 *points*. Line 3 in the file defines the X-values (i.e. points) shared by both of the sets (Y-values). In this case, each X-value corresponds to one of the columns of Quarterly Results. Lines 5 and 7 in *mm63.dat* define each of the sets of Y-values.

There are two types of XRT/graph data: `ARRAY` and `GENERAL`. Use `ARRAY` when the sets of Y-values share common X-values. Use `GENERAL` when the Y-values do not share common X-values, or when all sets do not have the same number of values.

1.6 Adding Header, Footer, and Labels

The *plot1.c* program is lacking in a few respects. There is no header, footer, or legend, and the X-axis numbering is not meaningful.

XRT/graph will always try to display a reasonable graph, even if very few resources have been specified. XRT/graph will use defaults for all unspecified resources.[2]

All resources for a particular graph may be specified when the graph widget is created. Resources may also be changed at any time during the execution of the

1. A more general way is to allocate a data object using the *XrtDataCreate()* procedure, and then to populate this object with data using C code.

2. The defaults calculated by XRT/graph can change quite frequently. For example, the default for `XtNxrtXNum` which specifies the X-axis numbering increment depends on the current graph area size (which can change when the window is resized), the value of `XtNxrtXNumMethod`, and possibly the current value of `XtNxrtXPrecision` (the X-axis numbering precision).

program with *XtSetValues()* or *XtVaSetValues()*. A programmer may ask for the current value of any resource with *XtGetValues()* or *XtVaGetValues()*.

To add a header, footer, legend, and meaningful X-axis labels, a few more resources will be added to the *XtVaCreateManagedWidget()* call. The following lines specify a header and footer to display above and below the graph:

```
XtNxrtHeaderStrings, header_strings,
XtNxrtFooterStrings, footer_strings,
```

Both of these resources take a `NULL`-terminated array of strings as their value. These must be defined elsewhere, for example:

```
static char *header_strings[] = { "Michelle's Microchips", NULL };
static char *footer_strings[] = { "1963 Quarterly Results", NULL };
```

By default, header, footer and legend strings are rendered in the "fixed" font. The header strings will be centered above the graph, while the footer strings will be centered below it. Also, by default, the legend is located to the right of the graph.

All array data used with XRT/graph should be thought of as *n sets* of *m points*. In this case, there are 2 sets of 4 points. Labels for each of the sets and each of the points can be specified as follows:

```
static char *set_labels[] = { "Expenses",
            "Revenue", NULL };
static char *point_labels[] = { "Q1", "Q2",
            "Q3", "Q4", NULL };
```

XRT/graph will use the Set-labels in the legend automatically. To use the Point-labels for the X-axis, we need to set `XtNxrtXAnnotationMethod` to `XRT_ANNO_POINT_LABELS`. Now the graph is created with a call that looks like:

```
graph = XtVaCreateManagedWidget("mygraph",
    xtXrtGraphWidgetClass,  toplevel,
    XtNxrtData,             my_data,
    XtNxrtHeaderStrings,    header_strings,
    XtNxrtFooterStrings,    footer_strings,
    XtNxrtSetLabels,        set_labels,
    XtNxrtPointLabels,      point_labels,
    XtNxrtXAnnotationMethod,XRT_ANNO_POINT_LABELS,
    XtNwidth,               500,
    NULL);
```

The resulting window is shown below in Figure 5.

The plot2 window is wider than plot1 because the Core resource `XtNwidth` is set to 500. To access this resource, we have also included the line:

```
#include <X11/StringsDefs.h>
```

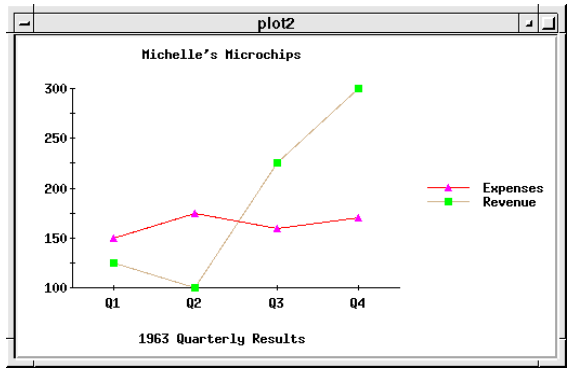


Figure 5 The plot2 Window

1.7 Changing to a Bar Chart

A powerful feature of XRT/graph is the ability to change a resource independently of any other resource.[1] For example, we can change the type of a graph from bar to plot to pie at any time.

The following line has been added to the *XtVaCreateManagedWidget()* call in *plot2.c* to make *bar2.c*:

```
XtNxrtType, XRT_TYPE_BAR,
```

When *bar2.c* is compiled and run, the window in Figure 6 displays.

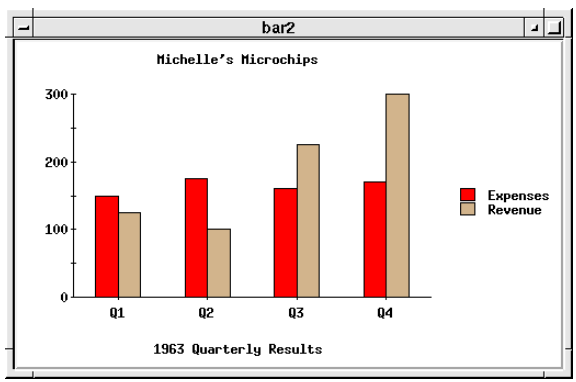


Figure 6 The bar2 Window

1. Although there are interdependencies between some resources, most resources are completely independant. Interdependencies are documented in Appendix A, *Resource Reference*.

1.8 Inverting Graph Orientation

Most graphs display the X-axis horizontally and the Y-axis vertically. It is often appropriate, however, to invert the sense of the X- and Y-axis.

In a plot, inverting causes the Y-values to be plotted against the horizontal axis, and the X-values to be plotted against the vertical. In a bar graph, it causes the bars to be displayed horizontally instead of vertically.

When programming XRT/graph, do not assume that the X-axis is always the horizontal axis. Determining which axis is vertical and which horizontal depends on the value of `XtNxrtInvertOrientation`.

XtNxrtInvertOrientation

To invert, set the `XtNxrtInvertOrientation` resource to `True`. By default, it is `False`.

Figure 7 shows the windows created by *plot2.c* and *bar2.c* with `XtNxrtInvertOrientation` set to `True`.

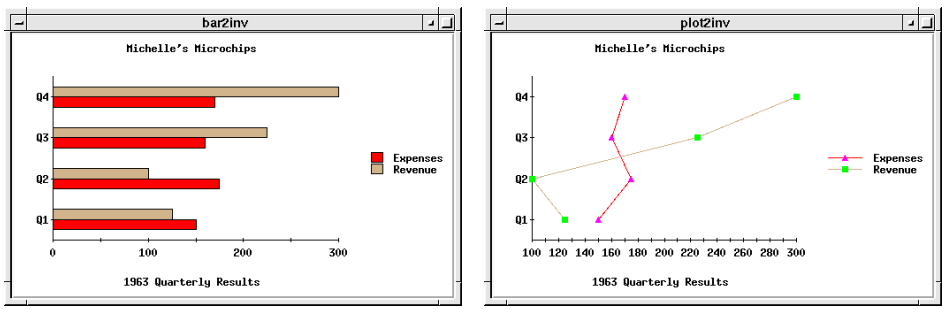


Figure 7 plot2 and bar2 windows with XtNxrtInvertOrientation True

1.9 Putting it all together

The *panel.c* program, located in *$XRTHOME/src/graph/examples* , brings together the concepts developed in this chapter. When *panel.c* is compiled and run, the window in Figure 8 displays.

The user can change how Michelle's Microchips data is displayed by clicking on one or more of the toggle widgets at the top of the window.

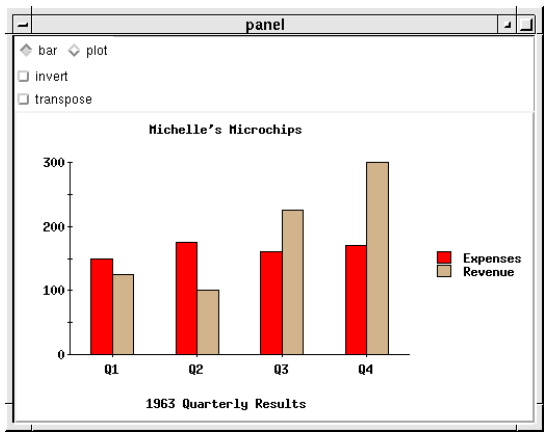


Figure 8 The panel Window

Each toggle widget has a callback procedure which changes the appropriate graph resource.

For example, the callback procedure for the invert toggle sets the `XtNxrtInvertOrientation` resource:

```
void
invert_proc(
    Widget          w,
    XtPointer       client_data,
    XtPointer       call_data
    )
{
    XmToggleButtonCallbackStruct *cbs;
    cbs = (XmToggleButtonCallbackStruct *) call_data;
    XtVaSetValues(graph,
        XtNxrtInvertOrientation,        cbs->set,
        NULL);
}
```

1.10 Sample Code

XRT/graph provides two types of programming examples. *Examples* illustrate *specific* features quickly and simply. *Demos* use *several* features in a richer, more complex application.

Examples

The following simple examples are located in *$XRTHOME/src/graph/examples*:

- *bar2tran.c* and *bar2tran_cpp.cxxbubble.c*
- *common.c*
- *panel.c*
- *pie.c*
- *plot1.c*
- *radar.c*
- *scrollgraph.c*
- *simple_uil.c*
- *slider.c*
- *stock.c*
- *time.c*

Demos

The following more complex demos are located in *$XRTHOME/src/graph/demos*. Each demo has its own directory.

- *builder* uses the XRT/graph Property Editor to provide a simple environment for interactively building the visual look of a graph or viewing the sample *.XRT* graph description files.
- *drill_down* shows one way to manage and present a large amount of data at different levels of granularity. It uses multiple graphs to display a list of previously-viewed graphs, the main graph, and the legend.
- *feedback* demonstrates some techniques for mapping and picking, and for displaying visual feedback of those events. It also illustrates one way to use markers.
- *flow* demonstrates a type of pulse chart, and is a good example of using color to differentiate ranges of data values. It uses two graph objects: one to plot the data, and another to show the legend. The legend is shown in its own pop-up window.
- *hilow* plots the open, high, low, and close prices for two stocks. It illustrates one way to manipulate data objects (on the fly) as a user requests various options.
- *polar* shows different ways to display polar data using the polar resources. It uses a number of mathematical functions to generate the data.
- *scroll_time* demonstrates usage of the time-axis features, and shows the use of *Xlib* routines to select a subset of data to display.

- *stripper* demonstrates how to use the Fast Update procedures to build a real-time strip chart. You can adjust the rate of incoming data and the scroll-back percentage.
- *text_area* demonstrates how to create and attach text areas to plots, bar graphs, stacking bar graphs, radar graphs and pie charts. Initial startup page displays background images.
- *vmgraph* displays your system's virtual memory statistics in three graphs that are updated every second. It demonstrates one way to read data from a pipe. This demo requires the *vmstat* program which is not available on some systems.

Multi-Product Demos

Several demos use XRT/graph with other XRT products. The following demos are located in their own directory in *$XRTHOME/src/common_demos*.

- *graph_editor* uses XRT/table to display graph data in a grid.
- *mvc* demonstrates the XRT MVC Kit, using XRT/graph, XRT/table, and XRT/3d.
- *profile* uses XRT/graph to provide 2D profiles of a 3D surface displayed by XRT/3d.

2

XRT/graph Basics

Integration with Motif ■ *Terminology*
Graph Types ■ *Floating Point Resources*
UseDefault Resources ■ *Pointer Resources*
String Resources ■ *Colors*
Using Resource Files

2.1 Integration with Motif

Conceptually, XRT/graph adds a new widget to the Motif toolkit. A Motif programmer manipulates this widget using the familiar Xt Intrinsics calls.

The Xt Intrinsics defines an object-oriented architecture that organizes widgets into classes. For example, the Motif XmPushButton widget is a subclass of XmLabel, which is a subclass of XmPrimitive, which is subclassed from the Xt Core widget. The XRT/graph widget is subclassed from the XmPrimitive widget. This means that graphs have XmPrimitive and Core resources that can be set in addition to the resources defined by the graph widget. For example, the `XmNshadowThickness` resource (defined by XmPrimitive) can be used to adjust the width of the border surrounding the graph. Also, the `XtNwidth` and `XtNheight` resources (defined by Core) can be used to adjust the width and height of the graph.

2.2 Terminology

Figure 10 shows the major components of the widget. There are four major areas in every graph: Header, Footer, Legend and Graph. Each area has its own origin, width and height. In Figure 10, each area has been drawn with a plain, 2-pixel-wide border.

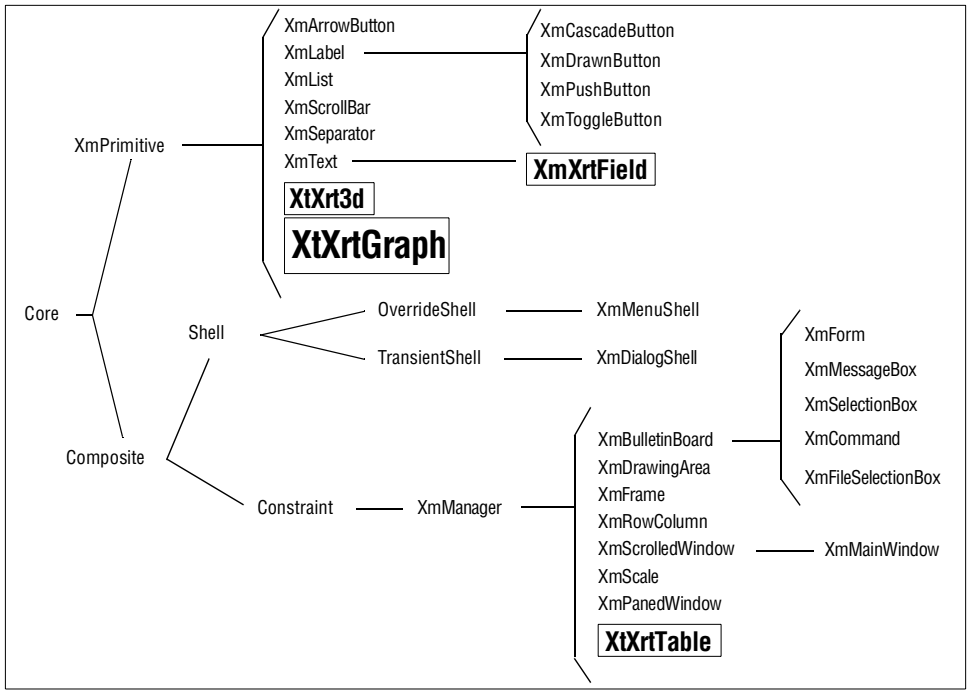


Figure 9 Graph Widget's Position within the Motif Class Hierarchy

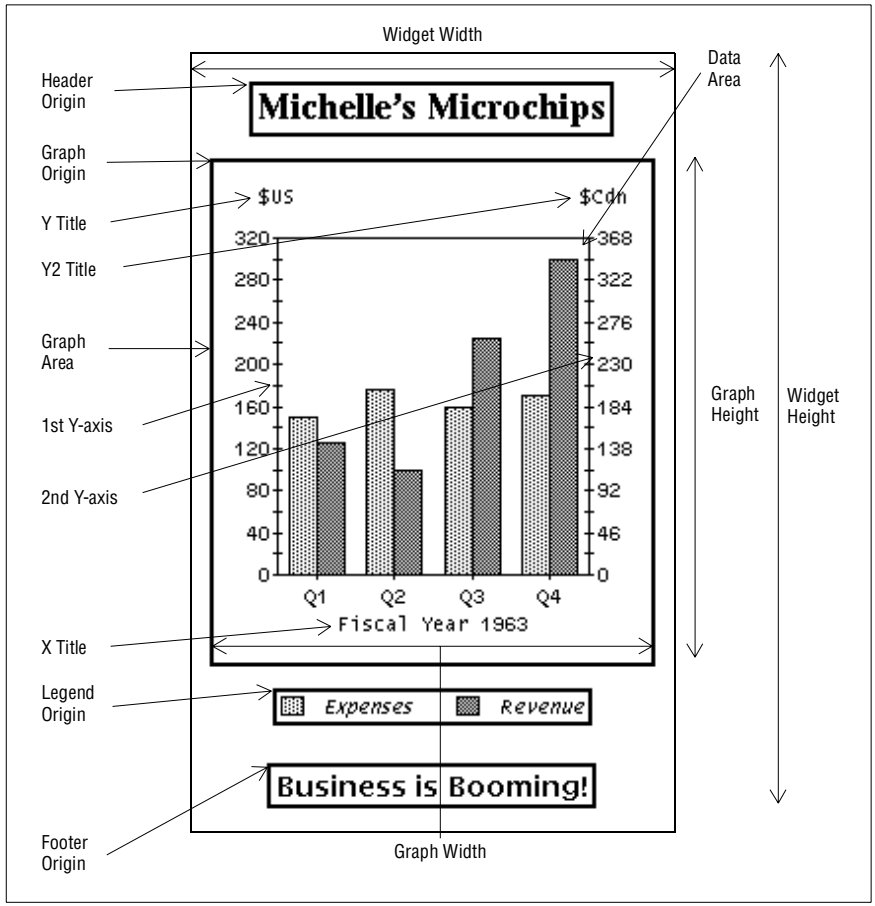
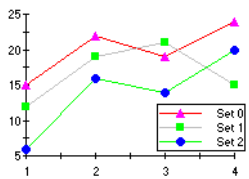
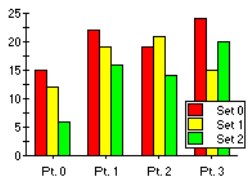
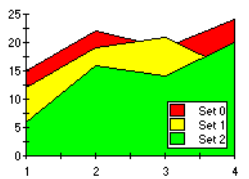
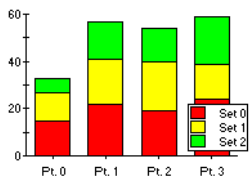


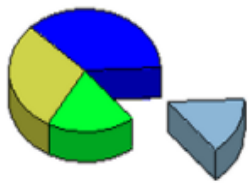
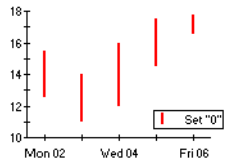
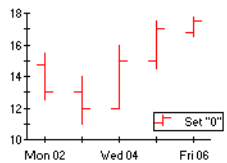
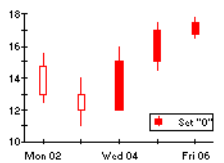
Figure 10 Graph Areas

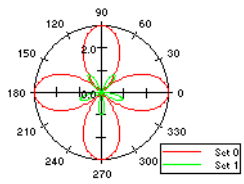
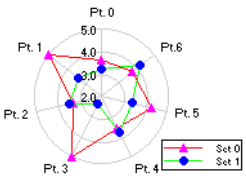
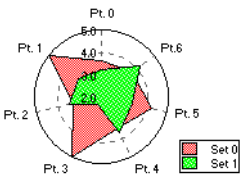
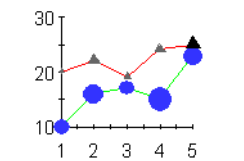
2.3 Graph Types

XRT/graph can display data as one of twelve basic graph types. These basic graph types can be used to build more specialized chart types, such as Pareto or Gantt charts.

The `XtNxrtType` resource specifies the graph type. This can be changed independently of the data or other resources, allowing you to easily display the same data as different graphs. The following table shows the basic graph types, the type of data allowed for each, and its basic characteristics.

	Example	Array data	General data	Notes
1	 XRT_TYPE_PLOT	✔	✔	**Plot** Draws each set as connected points of data. ■ When using Array data, X-values shared across sets ■ Set's appearance determined by `XrtDataStyle` line pattern, width, color, point style, point color, and point size elements ■ Stacking and 100% axis fill available when data source is an array
2	 XRT_TYPE_BAR	✔		**Bar** Draws each set as a bar in a cluster. The number of clusters is the number of points in the data. Each cluster displays the *n*th point in each set. ■ X-axis generally annotated using Point-labels ■ Set's appearance determined by `XrtDataStyle` fill pattern and color elements
3	 XRT_TYPE_AREA	✔	✔	**Area** Draws each set as connected points of data, filled below the points. Each set is drawn "on top" of the preceding set. ■ When using Array data, X-values shared across sets ■ Set's appearance determined by `XrtDataStyle` fill pattern and color elements ■ Stacking and 100% axis fill available when data source is an array
4	 XRT_TYPE_STACKING_BAR	✔		**Stacking Bar** Draws each set as a portion of a stacked bar cluster, the number of clusters being the number of data points. Each cluster displays the *n*th point in each set. Negative Y-values are stacked below the X-axis. ■ X-axis generally annotated using Point-labels ■ Set's appearance determined by `XrtDataStyle` fill pattern and color elements ■ 100% axis fill available

	Example	Array data	General data	Notes
5	 XRT_TYPE_PIE	✔		**Pie** Draws each set as a slice of a pie. The number of pies is the number of points in the data (values below a certain threshold can be grouped into an *other* slice). Any or all slices can be *exploded* to show emphasis. Each pie displays the *n*th point in each set. ■ Pies are annotated with Point-labels only ■ Set's appearance determined by `XrtDataStyle` fill pattern and color elements
6	 XRT_TYPE_HILO	✔		**HiLo** (see 4.12 on page 84 for details) Draws *two* sets together as a "High-low" bar. The points in each set make up one component of the bar: 1st set — points are the "high" value 2nd set — points are the "low" value ■ Each pair of sets forms one *logical* set, used with any function or resource[a] when specifying a set ■ Appearance determined by `XrtDataStyle` line pattern, width, and color elements in **first** set of the two
7	 XRT_TYPE_HILO_OPEN_CLOSE	✔		**HiLoOpenClose** (see 4.12 on page 84 for details) Similar to HiLo, but draws *four* sets together as a "High-low-open-close" bar. The additional sets' points make up the other components of the bar: 3rd set — points are the "open" value 4th set — points are the "close" value ■ Each 4 sets forms one *logical* set, used with any function or resource[a] when specifying a set ■ Appearance determined by `XrtDataStyle` line pattern, width, color and point size elements in **first** set of the four
8	 XRT_TYPE_CANDLE	✔		**Candle** (see 4.12 on page 84 for details) A special type of HiLoOpenClose chart, draws four sets together as a "candle" bar. ■ Each 4 sets forms one *logical* set, used with any function or resource[a] when specifying a set ■ *Simple* candle appearance determined by `XrtDataStyle` line pattern, fill pattern, width, color and point size elements in **first** set of the four ■ *Complex* candle appearance determined by different `XrtDataStyle` elements from **each** set of the four.

	Example	Array data	General data	Notes
9	 XRT_TYPE_POLAR	✔	✔	**Polar** (see 4.13 on page 87 for details) Draws each set as connected points of data on a polar coordinate system (*theta,r*). X-values represent the amount of rotation and Y-values are the distance from the origin. ■ When using Array data, X-values shared across sets ■ X-axis bounds cannot be set; Y-axis bounds cannot be set inside the data extents ■ Set's appearance determined by `XrtDataStyle` line pattern, width, color, point style and point size elements ■ Stacking and 100% axis fill available when data source is an array
10	 XRT_TYPE_RADAR	✔		**Radar** (see 4.13 on page 87 for details) Draws each set as connected points along radar "sticks" spaced equally apart. Each stick graphs the Y-value of the *n*th point in each set. ■ X-axis annotated with Point-labels or Value-labels ■ Set's appearance determined by `XrtDataStyle` line pattern, width, color, point style and point size elements ■ Stacking and 100% axis fill available
11	 XRT_TYPE_FILLED_RADAR	✔		**Filled Radar** (see 4.13 on page 87 for details) Draws each set as a connected points of data, filled inside the points. Each set is drawn "on top" of the preceding set. ■ X-axis annotated with Point-labels or Value-labels ■ Set's appearance determined by `XrtDataStyle` fill pattern and color elements ■ Stacking and 100% axis fill available
12	 XRT_TYPE_BUBBLE	✔		**Bubble** (see 4.12 on page 84 for details) Similar to a plot chart, but an extra piece of information is encoded varying either the diameter or area of the individual point symbols. ■ Logical set consists of two physical sets ■ Appearance determined by `XrtDataStyle` line pattern, width, color and point size elements in **first** set of the two

a. The only exceptions to this rule are the data values and data styles. Both use the actual set index, not the logical set, to allow customization of the values or appearance within a bar.

2.4 Floating Point Resources

When setting floating point resources using *XtSetValues()* or *XtVaSetValues()*, the *XrtFloatToArgVal()* function must be used.[1] For example, to double the current Y-axis maximum:

```
float   ymax;
XtVaGetValues(mygraph,
            XtNxrtYAxisMax, &ymax, NULL);
ymax = ymax * 2.0;
XtVaSetValues(mygraph,
    XtNxrtYAxisMax, XrtFloatToArgVal(ymax),
    NULL);
```

2.5 UseDefault Resources

Many XRT/graph resources have corresponding `UseDefault` resources, for example, `XtNxrtYNum` and `XtNxrtYNumUseDefault`.

`UseDefault` resources are Booleans that determine if XRT/graph should calculate a default value for the resource.

If, for example, `XtNxrtYNumUseDefault` is `True`, every time a graph is drawn, XRT/graph will go to considerable effort to determine a reasonable default value. It will consider the size of the graph area, the value of `XtNxrtYPrecision`, and possibly other factors. If, on the other hand `XtNxrtYNumUseDefault` is `False`, XRT/graph will use the last specified value of `XtNxrtYNum` as the `XtNxrtYNum` value.

Attempts to set a `UseDefault` resource to `False` will be ignored by XRT/graph unless the corresponding resource has been explicitly set or previously calculated by the widget. A side effect of setting any resource that has a corresponding `UseDefault` resource is that the `UseDefault` resource will be set to `False`.

The following code will force the value of `XtNxrtYNum` to whatever its current value is. It will also have the side effect of setting `XtNxrtYNumUseDefault` to `False`.

```
float   ynum;
XtVaGetValues(graph, XtNxrtYNum, &ynum, NULL);
XtVaSetValues(graph,
    XtNxrtYNum, XrtFloatToArgVal(ynum),
    NULL);
```

The following code will revert back to the default behavior, which is to have XRT/graph calculate a default value for `XtNxrtYNum` when it draws the graph.

```
XtVaSetValues(graph,
    XtNxrtYNumUseDefault,       True,
    NULL);
```

Attempts to set `UseDefault` resources in resource files are ignored by XRT/graph.

1. This function is not necessary when using *XrtSetValues()* to set floating point resources.

2.6 Pointer Resources

Many XRT/graph resources return pointers to structures or characters when used in an *XtGetValues()* call. A program should never use this pointer to change any data in memory. Everything the pointer points to should be considered "read-only". Convenience routines, such as *XrtDupDataStyles()* and *XrtDupStrings()* can be used to make copies of some types of data. These copies can be freed with *XrtFreeDataStyles()* and *XrtFreeStrings()*.

Graph data is the only exception to this rule. The `XtNxrtData` and `XtNxrtData2` resources should always reference data objects that exist in application memory.

Each graph makes its own copy of all data passed through a pointer resource (except for `XtNxrtData` and `XtNxrtData2`).

For example, if a program wants to interchange the first header string with the first footer string, the following code could be used:

```
char    **hs, **fs, **myhs, **myfs, *tmp;
XtVaGetValues(graph, /* Get read-only ptrs */
    XtNxrtHeaderStrings, &hs,
    XtNxrtFooterStrings, &fs,
    NULL);

myhs = XrtDupStrings(hs); /* Make copies */
myfs = XrtDupStrings(fs);

tmp = myfs[0];       /* Interchange 1st */
myfs[0] = myhs[0];
myhs[0] = tmp;

XtVaSetValues(graph, /* Reset them */
    XtNxrtHeaderStrings, myhs,
    XtNxrtFooterStrings, myfs,
    NULL);

XrtFreeStrings(myhs); /* Free my copies */
XrtFreeStrings(myfs);
```

2.7 String Resources

Several resources take strings or an array of strings as their values, for example, `XtNxrtXTitle` and `XtNxrtPointLabels`.

Be sure that only displayable characters are specified for these resources. Attempts to display strings containing ASCII control characters (such as '`\n`') may lead to unusual behavior.

If your application is using standard library routines such as *fgets()* to obtain its data, it will have to strip the trailing `\n` from each string before using the string as a value for an XRT/graph resource.

When setting a resource which takes an array of strings as its value, make sure that the last string pointer is `NULL`. XRT/graph looks for this `NULL` pointer to terminate the

list. Arrays of strings obtained from XRT/graph via `XtGetValues` will automatically be `NULL` terminated.

2.8 Colors

XRT/graph supports the specification of colors for the window background and foreground, as well as for the lines, fill patterns and points that represent data in the graph itself.

XRT/graph will choose default colors for the application, so simple applications need not concern themselves with color specification.

Color Specification

Colors are specified using a string which contains either the color name or the hexadecimal representation of the color's Red, Green and Blue (RGB) components. For example:

```
char * my_named_color = "LemonChiffon";
char * my_rgb_color = "#228B22";
```

If a color is specified by name, the name should appear in the X Window Server's color database. On Unix systems, this is normally the file */usr/lib/X11/rgb.txt*.

If the color's RGB value is specified instead of a named color, the string must begin with a '#' character followed by a two character hexadecimal value for each of the Red, Green and Blue components. In the example above, `my_rgb_color` is composed of 0x22 units of Red, 0x8B units of Green and 0x22 units of Blue.[1]

When an XRT/graph application's display is monochrome, all non-white colors will appear as black when used for the foreground or graph-area. All non-black colors appear as white when used for the background color.

Once a graph is created, an application should not update or change the application's colormap. Changes to any of the graph's colors should be made by updating the appropriate graph resources (i.e. `XtNxrtBackgroundColor`, `XtNxrtDataStyles`, etc.).

Colormap Flashing

X Window Server colormaps are a shared resource. In some situations, the number of colors in use by all the applications being displayed is more than the number of colormap entries available. In this case, the colors in some windows will "flash" to inappropriate colors as the user moves the mouse around the screen.

One way to eliminate color flashing is for the Server to provide only *static* colors. In this case, applications will not be given the exact colors they ask for, but will instead be given the closest color available within the window's colormap.

1. The syntax for color specification is not quite as restrictive as documented above. Any string acceptable to the Xlib procedure *XParseColor()* may be specified.

However, many applications require dynamic colors. In this case, each application will be given the exact colors it asks for[1]. Depending on the number of colors the Server has available, and the number and state of running applications, colormap flashing can occur.

When running on a Server with a dynamic colormap, XRT/graph applications will automatically allocate and de-allocate required color cells in the Server. This is done to minimize the XRT/graph application's demands on the colormap resource. If an XRT/graph application requests more colors than are available, black or white will be used instead of the unavailable color, unless color function has been provided by the user (see below).

Get Closest Color

XRT/graph uses strings rather than pixels for its color resources. The graph does its own color management using *XAllocColor()* and *XFreeColors()*. When the colormap runs out of colors, XRT/graph allocates either black or white, depending on the color the user is trying to allocate. The `XtNxrtColorFunc` resource enables developers to use their own color allocation scheme instead.

Along with the new resource, a sample color function (an adaptation of the XPM closest color matching algorithm, adapted to fit the semantics below) has been provided. You are free to use your own color function, as long as it follows the proper semantics. The source code for the sample color function is provided in *$XRTHOME/src/graph/util/x_color_func.c*. It is also included in the XRT/graph library as the method *XrtSampleColorFunction()* .

The `XtNxrtColorFunc` resource is `NULL` by default. In this case, it uses the normal black/white algorithm as before (the default behavior). If it is not `NULL`, it calls the supplied function and uses the color that it provides.

This user supplied function must have the following semantics to work with XRT/graph:

1. It must have the function prototype of the function below.
2. It must allocate a color (using *XAllocColor()*) because XRT/graph will use `XFreeColors` to free it when the widget is destroyed or when the user changes the color using *XtVaSetValues()*. If it cannot allocate a color, it should return `NO_PIXEL_VALUE`. In this case, XRT/graph will allocate black or white as it would if no color function was specified.
3. If it needs to allocate/free memory, as the sample color function does, it should do so (using the do-free flag) in the same manner as the sample color function.
4. If the color parameter is `NULL`, it should not allocate a color and return `NO_PIXEL_VALUE` (XRT/graph calls this function with color = `NULL` and do-free = `True` when the widget is destroyed).

1. Within the capabilities of the display's hardware.

Color function prototype:

```
unsigned long (*xrt_color_func)(
    Display     *display,
    int         screen,
    Colormap    colormap,
    XColor      *color,
    Boolean     do-free
    )
```

2.9 Using Resource Files

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 – The application end-users can modify the value of resources to suit their own needs.

With the exception of `UseDefault` resources, all XRT/graph resources can be set in resource files. However, most resources are too complex or too detailed for end-users to want to modify them.[1] Resource files for all the widgets in an application can be combined to create an application defaults file.

For further information on resource files and their usage, please see your Xt Intrinsics documentation. Appendix I on page 283 provides details on the syntax of resource settings when used in resource files, as well as a sample graph resource file.

1. If you use XRT/graph's Property Editor, you can use its Application Defaults dialog to individually select how each resource is to be defined; either in C code, or in a resource file.

3

Using the Property Editor

Getting Started ■ *Using the Property Editor*
Programming the Property Editor ■ *Generating Source Code*

The XRT/graph Property Editor is a separate library that developers can use to prototype, develop, and maintain XRT/graph applications. It can also be used to provide a customized graph property editor to end-users of your application. The library adds the following features to XRT/graph:

- Compact Property Editor that enables developers to interactively edit the graph resources of any graph in an application, as it runs.
- Ability to provide customized Property Editor to end-users as part of your application. You can control when users can invoke it, and whether users can load/save .XRT or data files.
- Save graph descriptions in a text file (called a .XRT file).
- Generate C language source code and resource files from .XRT files. The C code and resource files initialize a graph widget with the resources set to the values stored in the .XRT file.

The Property Editor library can save developers a lot of time. Graphs can be prototyped and shown to potential end-users without having to write any code. Developers can experiment with combinations of resource settings, seeing results immediately in the context of a running application, a great debugging aid.

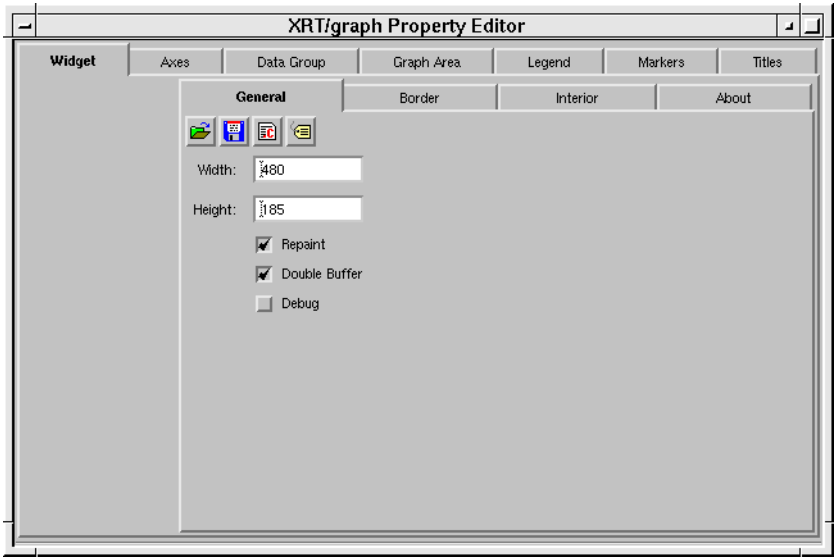


Figure 11 The XRT/graph Property Editor

3.1 Getting Started

The XRT/graph Property Editor can be accessed using the *builder* demo program or by adding it to your own program. Each method has its advantages:

- The *builder* demo provides a simple environment for building the visual look of a graph or viewing the sample .XRT graph description files.
- Access it from your program to help debug graph resource settings or experiment with different resources while your program runs.

Using Builder

Type "`builder`" in a terminal window to run the precompiled *builder* demo. It displays a window containing an XRT/graph widget (with a message in its header) and pops up the Property Editor. The source code for this demo is located in *$XRTHOME/src/graph/demos/builder*.

Using Your Application

The Property Editor library **must** be explicitly added to an application before any of its features can be used. Its implementation as a separate library makes it easy to use during development and remove from the finished application. To add the library and enable the Property Editor, perform the following steps:

1. Add the Property Editor (*libxrtmprop.a*), XRT/table, XRT/gear, XRT/field, and XPM libraries after the XRT/graph library in the list of libraries used when linking your application (see the Makefile for the *builder* demo).

Note that you do not need a license for these products to use the Property Editor in your application; ignore the no-license message from *xrt_auth*).

2. Call the *XrtPropertyEditorEnable()* procedure after creating a graph:

```
graph = XtVaCreateManagedWidget("graph",
    xtXrtGraphWidgetClass, toplevel,
    NULL);
...
XrtPropertyEditorEnable(graph, True, NULL, True, True);
```

3. Copy the fallback resources from the *builder* demo to your program. These resources control the size and colors of the Property Editor dialog.
4. Save, compile, and run your application.

Click the right mouse button on the graph widget to display the Property Editor.

3.2 Using the Property Editor

Click the right mouse button on a graph to display the Property Editor. Note that the application must enable the Property Editor for the graph and include the additional libraries it needs, as described in section 3.1 above.

Editing Graph Properties

1. Select the tab that corresponds to the element of the graph that you want to edit. Tabs contain one or more levels of inner tabs; find and select inner tabs until you see the property you want to edit.
2. If the tab contains a tree control on the left side, select the specific object to edit from that list.
3. Edit the property value.

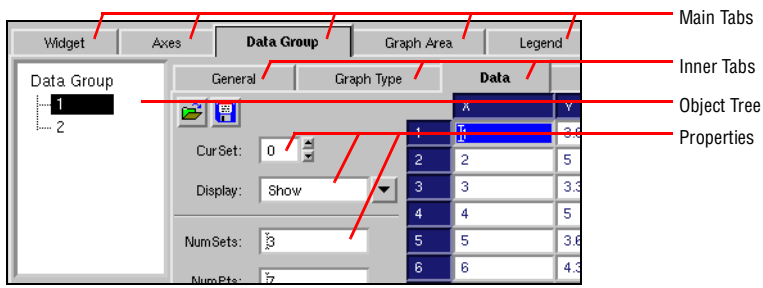


Figure 12 Elements of the Property Editor

As you edit resource values with the Property Editor, the changes are immediately applied to the graph and displayed.[1] This enables you to quickly see how a change affects the graph so that you can make further changes without leaving the Property

1. This assumes the Repaint property on the *Widget* tab's *General* inner tab is checked.

Editor. However, once you have changed a property the only way to "undo" it is to change the property back to its previous value.

Make further changes to the same property or change other properties on this tab or other tabs. To close the Property Editor, select "Close" from the Window manager control menu.

Loading and Saving .XRT Files

Use the toolbar buttons on the *Widget* tab's *General* inner tab to load or save a .XRT graph description. A collection of sample graphs is located in the *$XRTHOME/src/graph/builder* directory.

Loading and Saving Graph Data Files

Use the toolbar buttons on the *Data Group* tab's *Data* inner tab to load or save a data file. The data file must conform to the syntax understood by *XrtDataCreateFromFile()* (see Appendix D on page 215 for details). A collection of sample data files is located in the *$XRTHOME/src/graph/data* directory.

Limitations

The Property Editor provides access to all XRT/graph resources except text object attributes. Also, if the `XtNxrtData` resource points to the old Xrt Data Structure (pre XRT/graph 3.0), editing data is disabled.

3.3 Programming the Property Editor

The Property Editor library provides a number of procedures that enable you to use it in different ways. Applications using these procedures must link to the Property Editor library as described in section 3.1 on page 34.

3.3.1 Customizing End-User Property Editor

If you choose to provide the Property Editor to end-users of your application, you can easily disable and customize its capabilities using the *XrtPropertyEditorEnable()* procedure. *XrtPropertyEditorEnable()* has the following prototype:

```
Boolean
XrtPropertyEditorEnable()
    Widget      graph,
    Boolean     enable,
    char        *title_string,
    Boolean     allow_load_save,
    Boolean     do_updates
    )
```

To disable it at any time, pass *enable* as False. To customize the text that appears in the title bar of the Property editor, specify your title as the *title_string* argument. To disallow users from loading/saving .XRT or data files, specify *allow_load_save* as `False` (this also removes the KL Group logo from the *Widget* tab's *About* inner tab).

3.3.2 Popup, Popdown, and Update Programatically

An application can explicitly display, dismiss, or control updating of the Property Editor using several functions defined in the *XrtGraphProp.h* header file. To use these functions, an application must "`#include XrtGraphProp.h`" and link to the XRT/field, XRT/gear, and XRT/table libraries (as well as the XRT/graph and Property Editor library).

XrtPopupPropertyEditor()

Displays the Property Editor.

```
Boolean
XrtPopupPropertyEditor(
    Widget      graph,
    char        *title_string,
    Boolean     allow_load_save
    )
```

title_string is a custom text string to display in the title bar. *allow_load_save* specifies whether the user can load/save *.XRT* and data files and generate C code.

XrtPopdownPropertyEditor()

Closes the Property Editor. Does not destroy the dialog unless *destroy* is `True`.

```
Boolean
XrtPopdownPropertyEditor(
    Widget      graph,
    Boolean     destroy
    )
```

XrtUpdatePropertyEditor()

Updates the Property Editor with the current graph resource values.

```
Boolean
XrtUpdatePropertyEditor(
    Widget      graph
    )
```

3.3.3 Loading and Saving Graph Descriptions

Use the *XrtLoadGraphFromFile()* procedure to load a graph description and replace the current resource values. Use *XrtSaveGraphToFile()* to save the current resource values to a .XRT graph description file. These procedures have the following prototypes, defined in *XrtGraphInv.h*:

```
Boolean
XrtLoadGraphFromFile(
    Widget      graph,
    char        *filename,
    char        *msg
    )
```

```
Boolean
XrtSaveGraphToFile(
    Widget      graph,
    char        *filename,
    char        *msg
    )
```

filename is the pathname of the .XRT file. If any errors are encountered, they return `False`, and write an error message in *msg*. If *msg* is `NULL`, the error is written to *stderr*. *msg* should be at least 100 bytes.

To use these functions, an application must "`#include XrtGraphInv.h`" and link to the Property Editor library (*libxrtmprop.a*).

3.3.4 Property Editor Source Code

The Property Editor is provided as source code to enable developers to customize it for the needs of their applications. You can remove tabs or change how it works in any way. Note that a valid license of XRT PDS or licenses of XRT/table, XRT/gear, and XRT/field are required to compile a customized Property Editor.

The source code can be found in *$XRTHOME/src/graph/properties*.

3.4 Generating Source Code

Once you have created a .XRT file, you can generate C language source code that may be compiled into a working application or embedded in a more complex application. The source code creates a graph widget with all the resources set to the values defined in the .XRT file. Each resource will be set in either C code or in a resource file, depending on what was chosen in the Property Editor's *Application Defaults* properties dialog.

There are three ways to generate source code:

- by running the *xrt_gmf* program
- directly from the Property Editor, or
- programmatically from your application using *XrtGenerateCode()*.

xrt_gmf Program

To run *xrt_gmf*, enter "`xrt_gmf <filename>`", where `<filename>` is the name of a .XRT file. (You do not have to type the ".XRT" part of the filename.) *xrt_gmf* will output messages as it executes. For example:

```
% xrt_gmf mygraph
Reading mygraph.XRT
Writing mygraph_xrt.c and mygraph_xrt.res
Writing mygraph_xrt.h
Writing mygraph_xrt_main.c
Writing mygraph_xrt.make
```

To generate code to be used in ANSI C or C++ applications, specify *-a* when running *xrt_gmf*.

Using the Property Editor

Click the buttons shown below on the *Widget* tab's *General* inner tab to generate Application defaults and code for the current graph:



The Property Editor only generates ANSI C code.

XrtGenerateCode() Procedure

The *XrtGenerateCode()* procedure generates source code files from a specified .XRT file. It has the following prototype, defined in *XrtGraphInv.h*:

```
Boolean
XrtGenerateCode(
    Widget      graph,
    char        *filename,
    Boolean     gen_ansi,
    char        *msg
    )
```

filename is the pathname of the .XRT file. *gen_ansi* specifies whether to generate ANSI C code. If *XrtGenerateCode()* encounters any errors, it returns `False`, and writes an error message in *msg*. If *msg* is `NULL`, the error is written to *stderr*. *msg* should be at least 100 bytes.

Remember to include the *XrtGraphInv.h* header file.

3.4.1 Files Created by Source Code Generator

Generated source files should not be edited by hand because they will be overwritten every time the source code generator is run.

The _xrt.c File

The *_xrt.c* file defines a procedure that creates the graph widget. It sets the graph resources that should be defined in C code (defined using the Property Editor's *Application Defaults* dialog).

This file can be linked into larger applications to embed the graph it defines into the application. The name of this procedure depends on the graph name. For example, if the .XRT file is named *mygraph.XRT*, then the *mygraph_xrt.c* source file defines the following procedure:

```
Widget
mygraph_xrt_create(parent)
Widget      parent;
{
    /* Create graph & set resources etc. ... */
}
```

parent is the handle of the widget that will manage the graph. *mygraph_xrt_create()* returns the new graph widget handle.

The _xrt.res File

The *_xrt.res* file is a Motif resource file[1] containing the default resource values for the graph widget. Only the resources identified as resource-file defaults are defined here (using the Property Editor's *Application Defaults* dialog). The rest are defined in the *_xrt.c* file.

The _xrt.h File

The *_xrt.h* file contains declarations required by source code files that call the procedures in the *_xrt.c* file. Include this file in application source code files that call the *_xrt_create()* routine. For example:

```
#include "mygraph_xrt.h"
```

The _xrt.make File

The *_xrt.make* file is a sample Makefile which directs the *make(1)* command on how to compile a sample application from the files created with the generated files. You may want to rename this sample Makefile and use it as a template for your application Makefile.

The _xrt_main.c File

The *_xrt_main.c* file is a sample mainline C program that:

- initializes the Xt Intrinsics and creates a top level widget
- creates a data object and loads it using *XrtDataCreateFromFile()*
- using the *_xrt_create()* procedure defined in the *_xrt.c* file, creates a graph managed by the top level
- attaches the data object to the graph widget, and
- calls *XtRealizeWidget()* and *XtMainLoop()*.

This sample mainline source may be renamed and used as a starting point for a more sophisticated XRT/graph application.

1. Resource files provide a very flexible way of defining widget resource values. For example, this file can be combined with other resource files to create an application defaults resource file. The application resource file may be installed in */usr/lib/X11/app-defaults*. Users can change a specific resource value in their own *.Xdefaults* file. See your Xt Intrinsics documentation for details about resource files.

3.4.2 Compiling and Running Generated Code

To compile the generated source files, enter "`make -f`" followed by the name of the generated Makefile. For example, if the Makefile is named *mygraph_xrt.make*, enter:

```
make -f mygraph_xrt.make
```

You will see the output of the compiler and linker as the source files are compiled into an executable. The executable name ends in "_xrt", e.g. *mygraph_xrt*. Once an executable is created, the makefile calls *xrt_auth* which checks for a valid XRT/graph license, and authorizes the executable. To run the compiled program, enter its name.

If you used the Application Defaults Properties window to specify resources to be written to the *_xrt.res* resource file, these resources must be loaded into the resource database prior to running the application. An easy way to do this while developing your application is to enter:

```
setenv XENVIRONMENT mygraph_xrt.res
```

4

Programming XRT/graph

Resource Summary ■ *Axis Labelling and Annotation Methods*
Axis Controls ■ *Annotation Formatting*
Positioning Graph Areas ■ *3D Effect* ■ *Stacking*
Header, Footer and Legend Display ■ *Data Styles*
Special Bar Chart Resources ■ *Special Pie Chart Resources*
HiLo, Candle and Bubble Graphs ■ *Special Polar/Radar Graph Resources*
Foreground and Background Colors ■ *Markers*
Area Borders ■ *Background Images* ■ *Double Buffering*
Adding Scrollbars to a Graph ■ *Output and Printing*

4.1 Resource Summary

This section summarizes all of the XRT/graph resources. It is not necessary to remember all the resources in order to program XRT/graph effectively. For most graphs, many resources may be left with their default settings.

The use of most resources is described in this chapter. Appendix A on page 171 contains a reference of all XRT/graph resources.

Resource	Type
XtNxrtData	XrtData *
XtNxrtData2	XrtData *
XtNxrtFrontDataset	int
XtNxrt[XYY2]Max	float
XtNxrt[XYY2]MaxUseDefault	Boolean
XtNxrt[XYY2]Min	float
XtNxrt[XYY2]MinUseDefault	Boolean

Figure 13 Data Resources

Resource	Type
XtNxrtDataAreaBackgroundColor	char *
XtNxrtDataAreaForegroundColor	char *
XtNxrtDataAreaImage	XrtImage
XtNxrtDataAreaImageLayout	XrtImageLayout (enum)
XtNxrtGraph3dShading	XrtShading (enum)
XtNxrtGraphBackgroundColor	char *
XtNxrtGraphBorder	XrtBorder (enum)
XtNxrtGraphBorderWidth	Dimension
XtNxrtGraphDepth	int
XtNxrtGraphForegroundColor	char *
XtNxrtGraphHeight	Dimension
XtNxrtGraphHeightUseDefault	Boolean
XtNxrtGraphImage	XrtImage
XtNxrtGraphImageLayout	XrtImageLayout (enum)
XtNxrtGraphInclination	int
XtNxrtGraphMarginBottom	Position
XtNxrtGraphMarginBottomUseDefault	Boolean
XtNxrtGraphMarginLeft	Position
XtNxrtGraphMarginLeftUseDefault	Boolean
XtNxrtGraphMarginRight	Position
XtNxrtGraphMarginRightUseDefault	Boolean
XtNxrtGraphMarginTop	Position
XtNxrtGraphMarginTopUseDefault	Boolean
XtNxrtGraphRotation	int
XtNxrtGraphShowOutlines	Boolean
XtNxrtGraphWidth	Dimension
XtNxrtGraphWidthUseDefault	Boolean
XtNxrtGraph[XY]	Position
XtNxrtGraph[XY]UseDefault	Boolean
XtNxrtType	XrtType (enum)
XtNxrtType2	XrtType (enum)
XtNxrtIsStacked	Boolean
XtNxrtIsStacked2	Boolean

Figure 14 Graph Resources

Resource	Type
XtNxrtPointLabels	char * *
XtNxrtPointLabels2	char * *
XtNxrtSetLabels	char * *
XtNxrtSetLabels2	char * *
XtNxrtTimeBase	time_t
XtNxrtTimeFormat	char *
XtNxrtTimeFormatUseDefault	Boolean
XtNxrtTimeUnit	XrtTimeUnit (enum)
XtNxrt[XYY2]AnnotationMethod	XrtAnnoMethod (enum)
XtNxrt[XYY2]AxisLabelFormat	char *
XtNxrt[XYY2]Labels	XrtValueLabel * *

Figure 15 Labeling Resources

Resource	Type
XtNxrtMarkerDataset	int
XtNxrtMarkerDataStyle	XrtDataStyle *
XtNxrtMarkerDataStyleUseDefault	Boolean
XtNxrt[XY]Marker	float
XtNxrtXMarkerMethod	XrtMarkerMethod (enum)
XtNxrtXMarkerSet	int
XtNxrtXMarkerPoint	int
XtNxrt[XY]MarkerDataStyle	XrtDataStyle *
XtNxrt[XY]MarkerDataStyleUseDefault	Boolean
XtNxrt[XY]MarkerShow	Boolean

Figure 16 Marker Resources

Resource	Type
XtNxrtDataStyles	XrtDataStyle * *
XtNxrtDataStyles2	XrtDataStyle * *
XtNxrtDataStylesUseDefault	Boolean
XtNxrtDataStyles2UseDefault	Boolean

Figure 17 Data Style Resources

Resource	Type
XtNxrtAxisBoundingBox	Boolean
XtNxrtAxisFont	XmFontList
XtNxrtInvertOrientation	Boolean
XtNxrt[XYY2]AnnoPlacement	XrtAnnoPlacement (enum)
XtNxrt[XYY2]AnnotationRotation	XrtRotate (enum)
XtNxrt[XYY2]AxisMax	float
XtNxrt[XYY2]AxisMaxUseDefault	Boolean
XtNxrt[XYY2]AxisMin	float
XtNxrt[XYY2]AxisMinUseDefault	Boolean
XtNxrt[XYY2]AxisShow	Boolean
XtNxrt[XYY2]AxisLogarithmic	Boolean
XtNxrt[XYY2]AxisReversed	Boolean
XtNxrt[XYY2]Grid	float
XtNxrt[XYY2]GridUseDefault	Boolean
XtNxrt[XYY2]GridDataStyle	XrtDataStyle *
XtNxrt[XYY2]GridDataStyleUseDefault	Boolean
XtNxrt[XYY2]Num	float
XtNxrt[XYY2]NumUseDefault	Boolean
XtNxrt[XYY2]NumMethod	XrtNumMethod (enum)
XtNxrt[XYY2]Origin	float
XtNxrt[XYY2]OriginUseDefault	Boolean
XtNxrt[XYY2]OriginPlacement	XrtOriginPlacement (enum)
XtNxrt[XYY2]Precision	int
XtNxrt[XYY2]PrecisionUseDefault	Boolean
XtNxrt[XYY2]Title	char *
XtNxrt[XYY2]TitleRotation	XrtRotate (enum)
XtNxrt[XYY2]Tick	float
XtNxrt[XYY2]TickUseDefault	Boolean
XtNxrtYAxisConst	float
XtNxrtYAxisMult	float
XtNxrt[YY2]Axis100Percent	Boolean
XtNxrt[XYY2]AxisDataStyle	XrtDataStyle *
XtNxrt[XYY2]AxisDataStyleUseDefault	Boolean

Figure 18 Axis Resources

Resource	Type
XtNxrtBarClusterOverlap	int
XtNxrtBarClusterWidth	int

Figure 19 Bar Resources

Resource	Type
XtNxrtOtherDataStyle	XrtDataStyle *
XtNxrtOtherDataStyleUseDefault	Boolean
XtNxrtOtherLabel	char *
XtNxrtPieMergeMissingSlices	Boolean
XtNxrtPieMinSlices	int
XtNxrtPieOrder	XrtPieOrder (enum)
XtNxrtPieThresholdMethod	XrtPieThresholdMethod (enum)
XtNxrtPieThresholdValue	float
XtNxrtPieStartAngle	float

Figure 20 Pie Resources

Resource	Type
XtNxrtCandleComplex	Boolean
XtNxrtHiLoCloseShow	Boolean
XtNxrtHiLoOpenShow	Boolean
XtNxrtHiLoOpenCloseFullWidth	Boolean
XtNxrtBubbleMethod	XrtBubbleMethod (enum)
XtNxrtBubbleMin	float
XtNxrtBubbleMax	float

Figure 21 HiLo/Candle/Bubble Resources

Resource	Type
XtNxrtAngleUnit	XrtAngleUnit (enum)
XtNxrtPolarHalfRange	Boolean
XtNxrtXOriginBase	float
XtNxrt[YY2]AnnotationAngle	float
XtNxrt[YY2]AnnotationAngleUseDefault	Boolean
XtNxrtPolarAxisAllowNegatives	Boolean

Figure 22 Polar/Radar Resources

Resource	Type
XtNxrtHeaderAdjust	XrtAdjust (enum)
XtNxrtHeaderBackgroundColor	char *
XtNxrtHeaderBorder	XrtBorder (enum)
XtNxrtHeaderBorderWidth	Dimension
XtNxrtHeaderFont	XmFontList
XtNxrtHeaderForegroundColor	char *
XtNxrtHeaderHeight	Dimension
XtNxrtHeaderImage	XrtImage
XtNxrtHeaderImageLayout	XrtImageLayout (enum)
XtNxrtHeaderImageMinimumSize	Boolean
XtNxrtHeaderStrings	char * *
XtNxrtHeaderWidth	Dimension
XtNxrtHeader[XY]	Position
XtNxrtHeader[XY]UseDefault	Boolean
XtNxrtFooterAdjust	XrtAdjust (enum)
XtNxrtFooterBackgroundColor	char *
XtNxrtFooterBorder	XrtBorder (enum)
XtNxrtFooterBorderWidth	Dimension
XtNxrtFooterFont	XmFontList
XtNxrtFooterForegroundColor	char *
XtNxrtFooterHeight	Dimension
XtNxrtFooterImage	XrtImage
XtNxrtFooterImageLayout	XrtImageLayout (enum)
XtNxrtFooterImageMinimumSize	Boolean
XtNxrtFooterStrings	char * *
XtNxrtFooterWidth	Dimension
XtNxrtFooter[XY]	Position
XtNxrtFooter[XY]UseDefault	Boolean

Figure 23 Header and Footer Resources

Resource	Type
XtNxrtLegendAnchor	XrtAnchor (enum)
XtNxrtLegendBackgroundColor	char *
XtNxrtLegendBorder	XrtBorder (enum)
XtNxrtLegendBorderWidth	Dimension
XtNxrtLegendFont	XmFontList
XtNxrtLegendForegroundColor	char *
XtNxrtLegendHeight	Dimension
XtNxrtLegendImage	XrtImage
XtNxrtLegendImageLayout	XrtImageLayout (enum)
XtNxrtLegendOrientation	XrtAlign (enum)
XtNxrtLegendReversed	Boolean
XtNxrtLegendShow	Boolean
XtNxrtLegendTitle	char *
XtNxrtLegendWidth	Dimension
XtNxrtLegend[XY]	Position
XtNxrtLegend[XY]UseDefault	Boolean

Figure 24 Legend Resources

Resource	Type
XtNxrtExposeCallback	XtCallbackList
XtNxrtMapCallback	XtCallbackList
XtNxrtModifyCallback	XtCallbackList
XtNxrtPickCallback	XtCallbackList
XtNxrtResizeCallback	XtCallbackList
XtNxrtRotateCallback	XtCallbackList
XtNxrtTransformCallback	XtCallbackList
XtNxrtZoomAxisCallback	XtCallbackList

Figure 25 Callback Resources

Resource	Type
XtNxrtBackgroundColor	char *
XtNxrtBorder	XrtBorder (enum)
XtNxrtDebug	Boolean
XtNxrtDoubleBuffer	Boolean
XtNxrtExtraDefaultDataStyles	Boolean
XtNxrtForegroundColor	char *
XtNxrtImage	XrtImage
XtNxrtImageLayout	XrtImageLayout (enum)
XtNxrtRepaint	Boolean

Figure 26 Other Resources

4.2 Axis Labelling and Annotation Methods

A variety of resources combine to determine the annotation that appears on the axes and the text displayed in the legend. The XtNxrt[XYY2]AnnotationMethod resources specify the method used to annotate each axis. The valid annotation methods are:

XRT_ANNO_VALUES (default)	The graph chooses appropriate axis annotation automatically, based on the graph type and the data itself.
XRT_ANNO_POINT_LABELS (X-axis only)	The graph spaces the points based on the X-values and annotates them with text you specify for each point.
XRT_ANNO_VALUE_LABELS	The graph annotates the axis with text you define for specific X- or Y-axis coordinates.
XRT_ANNO_TIME_LABELS (X-axis only)	The graph interprets the X-values as units of time along the X-axis, automatically choosing annotation based on the starting point and format you specify.

Point-labels and Time-labels annotation can only be used with the X-axis.

For each annotation method, XRT/graph makes use of some axis resources and ignores others. The following sections describe how to use each type of annotation.

4.2.1 Values Annotation

When XtNxrt[XYY2]AnnotationMethod is XRT_ANNO_VALUES, XRT/graph uses several resources to control the numeric values annotating the axis. Each resource controls one aspect of the annotation.

Numbering Method

XtNxrt[XYY2]NumMethod specifies the method used to number the axis when annotation method is XRT_ANNO_VALUES. Valid values are:

XRT_NUM_ROUND (default)	Axis numbering is rounded to values whose most-significant digit ends in 1, 2, or 5. It uses the value of XtNxrt[XYY2]Precision to format the annotation.
XRT_NUM_PRECISION	Axis numbering is calculated using the value of XtNxrt[XYY2]Precision, as well as XtNxrt[XYY2]Num and XtNxrt[XYY2]Tick.

When XRT_NUM_ROUND, XRT/graph numbers the axes by analyzing the data to be graphed, determining numbering increment, ticking increment, and appropriate numbering precision. Then it rounds the numbering and draws the axis. This method usually provides the most pleasing numbering.

When XRT_NUM_PRECISION, numbering is done the same as described above, except the numbering is not rounded.

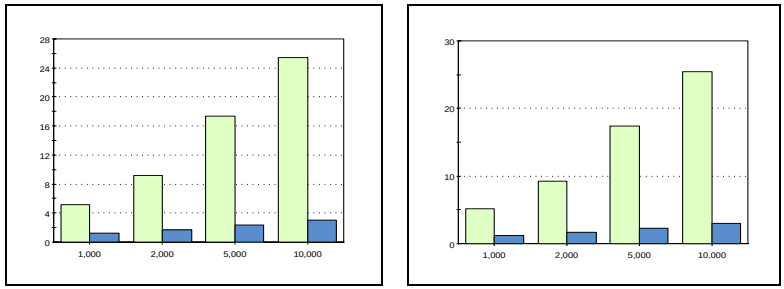


Figure 27 "Precision" and "Rounded" Axis Numbering

Precision

XtNxrt[XYY2]Precision serves two purposes, depending on the axis numbering method used:

- If XtNxrt[XYY2]NumMethod is XRT_NUM_ROUND, it specifies only the formatting of the axis numbering.
- If XtNxrt[XYY2]NumMethod is XRT_NUM_PRECISION, it in large part controls the axis numbering.

Positive values of XtNxrt[XYY2]Precision indicate the number of places after the decimal place. Negative values indicate the (positive) number of zeros to use before the decimal place.

For example, if XtNxrtXPrecision is -3 and XtNxrtYPrecision is 2, the X-axis numbering will be multiples of 1000 and the Y-axis numbering will be shown to 2 decimal places.

When XtNxrt[XYY2]PrecisionUseDefault is True, XRT/graph determines appropriate precision at runtime.

Numbering Increment

The XtNxrt[XYY2]Num resource is used to specify the interval between annotations along the axis. For example, if XtNxrtYNum is 3.2, XRT/graph will annotate every 3.2 units along the Y-axis, starting at the origin.

Tick marks along the axis are related to the numbering increment. Ticking increment is discussed in section 4.3.3 on page 62.

Be careful when explicitly setting the numbering (as well as the ticking and grid) increment resources. If they are set too small, graphs or axes may be illegible. To allow XRT/graph to determine appropriate numbering and tick increments, set the UseDefault resources to True.

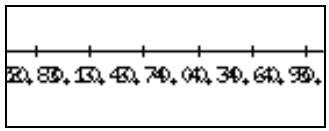


Figure 28 Axis with Numbering Increment Set too Small

Annotation Formatting

Axis annotation can be displayed in a variety of custom numeric, accounting, currency, percentage or scientific formats using the `XtNxrt[XYY2]AxisLabelFormat` resources. For complete details, see section 4.4 on page 66.

Logarithmic Axes

To convert from a linear to a $\log_{10}$ axis, set `XtNxrt[XYY2]AxisLogarithmic` to `True`. Since logarithmic axes are only capable of plotting values greater than zero, any data values less than or equal to zero are treated as holes (that is, as missing data values). It is not possible to set the numbering increment, ticking increment, or precision for a logarithmic axis. Axis or data minimum/maximum or origin cannot be set less than or equal to zero.

Several additional criteria must be met for a logarithmic X-axis:

- The graph type must be either plot, area, HiLo, HiLoOpenClose, or candle.
- The X-axis annotation method cannot be `XRT_ANNO_TIME_LABELS`.

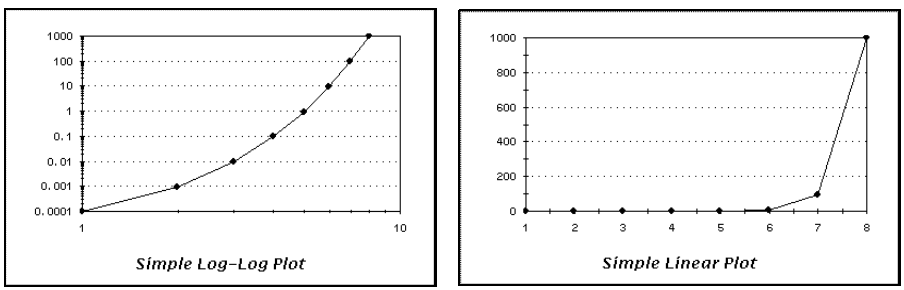


Figure 29 Linear and Logarithmic Axes

4.2.2 Point-labels Annotation and Set-labels

When `XtNxrtXAnnotationMethod` is `XRT_ANNO_POINT_LABELS`, XRT/graph uses Point-labels and Set-labels to label the rows and columns of Array data. Point-labels annotate the X-axis and Set-labels appear in the legend. For General data, Set-labels are used to label the various *lines* of data.[1]

1. Point-labels are not meaningful with General data because each set can have its own X-values and number of points.

Consider the *pop.dat* file listed in Figure 30. It is in a form suitable for loading with *XrtDataCreateFromFile()*. It defines a 3 by 5 array of population values for 3 cities.

```
ARRAY 3 5

# X Samples (One for the beginning of each decade 1960 to 2000)
1 2 3 4 5
# Mexico
5 8 14 19 24
# Tokyo
11 15 18 21 22
# New York
14 16 15.5 15.2 15.4
```

Figure 30 The pop.dat File

Using XRT/graph terminology, there are "3 *sets* of 5 *points*". The Set-labels are the city names, and the Point-labels are the year names. When this file is used to create a data object called `pop`, the following code can be used to define the labels (the result is shown in Figure 31):

```
static char *sl[] = { "Mexico City", "Tokyo",
                      "New York", NULL };
static char *pl[] = { "60", "70", "80", "90",
                      "2000", NULL };
popg = XtVaCreateManagedWidget("graph",
    xtXrtGraphWidgetClass, parent_widget,
    XtNxrtXAnnotationMethod, XRT_ANNO_POINT_LABELS,
    XtNxrtData,          pop,
    XtNxrtSetLabels      sl,
    XtNxrtPointLabels,   pl,
    NULL);
```

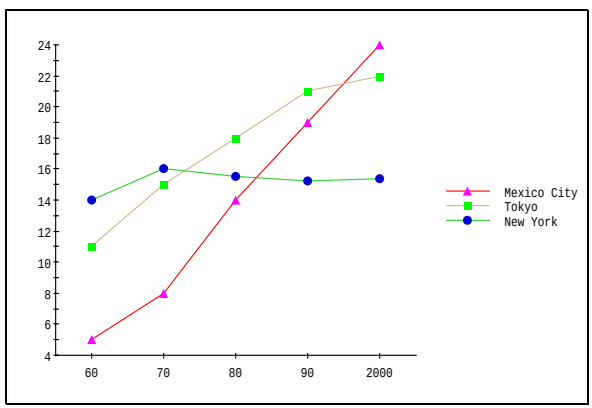


Figure 31 Point-label X-axis Annotation and Set-label Legend Text

As an alternative to using the XtNxrtSetLabels and XtNxrtPointLabels resources, individual labels can be accessed with the *SetNth* and *GetNth* methods, which are documented in Appendix D on page 211. For example:

```
/* Note that index is zero-based */
printf("Third Set-label is %s\n",
    XrtGetNthSetLabel(graph, 2) );

/* Make 3rd Point-label "eighty" */
XrtSetNthPointLabel(graph, 2, "eighty");
```

If the data for a graph is in XRT_DATA_GENERAL format, Point-labels are ignored.

4.2.3 Value-labels Annotation

When XtNxrt[XYY2]AnnotationMethod is XRT_ANNO_VALUE_LABELS, XRT/graph uses text strings you specify to label the axis. Use "Value-labels" to label significant axis values. Each Value-label is defined by an XrtValueLabel structure:

```
typedef struct {
    float   value;
    char    *string;
} XrtValueLabel;
```

value defines the X-, Y-, or Y2-value at which to display the label, and *string* is the text to display. An entire array of Value-labels can be programmed for an axis using XtNxrt[XYY2]Labels. Alternatively, the *XrtSetValueLabel()* and *XrtGetValueLabel()* procedures can be used to access individual Value-labels.

Value-labels should be used whenever it is important to honor the data's X-or Y-values, and custom axis annotation is required. For example, to put a custom Value-label of "foobar" on the X-axis at X=7.43, the following code could be used:

```
XrtValueLabel   mylabel;
my_label.value = 7.43;
my_label.string = "foobar";
XrtSetValueLabel(graph, XRT_AXIS_X, &mylabel);
```

The *XrtGetValueLabel()* method takes a pointer to an XrtValueLabel structure, and returns a pointer to another XrtValueLabel structure. The returned structure is filled in with the value-string pair of the Value-label that is closest to the value in the structure that was passed in. If there are no Value-labels, it returns NULL. The following example finds the closest Value-label to X=5.0:

```
XrtValueLabel   my_label, *closest_label;

my_label.value = 5.0;
closest_label = XrtGetValueLabel(graph, XRT_AXIS_X, &my_label);
if (closest_label != NULL) {    /* all parameters valid */
    if (closest_label->string == NULL)
        printf("No X-axis Value-labels exist!\n");
    else
        printf("Closest is at %f, text is %s\n",
            closest_label->value,
            closest_label->string);
}
```

To clear all Value-labels for an axis, set XtNxrt[XYY2]Labels to NULL.

To delete a particular Value-label, set its *string* to NULL. Beware of floating-point round-off errors when deleting Value-labels. The best way to delete a Value-label is to first get it, set the string to NULL, and then set it again. The following example deletes the label closest to X=5.0:

```
XrtValueLabel   label, *label_ptr;

label.value = 5.0;
label_ptr = XrtGetValueLabel(graph, XRT_AXIS_X, &label);
if (label_ptr != NULL) {        /* all parameters valid */
    if (label_ptr->string == NULL)
        printf("No labels to delete!\n");
    else {
        label = *label_ptr;
        label.string = NULL;
        XrtSetValueLabel(graph, XRT_AXIS_X, &label);
    }
```

Common uses for Value-labels include annotating particular dates on the X-axis and replacing Y-values with more meaningful text. Figure 32 shows an example of Value-labels on both axes.

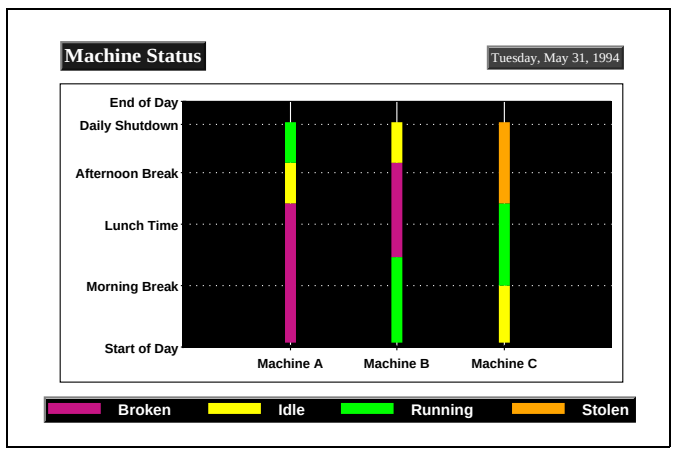


Figure 32 Using Value-labels to Annotate axes

4.2.4 Time-axis Labels Annotation

XRT/graph provides special support for an X-axis that represents the time dimension. Resources that are used to specify and control a time-axis are:

`XtNxrtTimeBase`	Specifies the moment in time your X-values begin measuring.
`XtNxrtTimeUnit`	Specifies the time units the X-values are measured in.
`XtNxrtTimeFormat`	Specifies the formatting style used for annotating the time-axis.
`XtNxrtTimeFormatUseDefault`	Specifies that the time format should be calculated dynamically.
`XtNxrtXAnnotationMethod`	Specifies the method used for annotating the X-axis. Set this to `XRT_ANNO_TIME_LABELS` for Time-labels.

Time-axis Criteria

The time-axis is ideal for applications that graph something measured in seconds, minutes, hours, days, weeks, months or years. Several criteria must be met before you can use a time-axis:

- The graph type cannot be pie, polar, radar, or filled radar – any other type is valid.
- The time range represented by the axis must be greater than the beginning of the year 1970 and less than the year 2038.
- The annotation resolution desired must not be less than one second.

If your needs do not meet these criteria, you will have to use another method for annotating the X-axis, such as Value-labels.

Time Units

When you set `XtNxrtXAnnotationMethod` to `XRT_ANNO_TIME_LABELS`, XRT/graph will interpret the X-values in your data using time units. Seconds are the default time unit, so if your X-values range from a low of 20 to a high of 55, XRT/graph will draw an X-axis that spans 35 seconds.[1]

You can set a different time unit using the `XtNxrtTimeUnit` resource. Valid values are: `XRT_TMUNIT_SECONDS`, `XRT_TMUNIT_MINUTES`, `XRT_TMUNIT_HOURS`, `XRT_TMUNIT_DAYS`, `XRT_TMUNIT_WEEKS`, `XRT_TMUNIT_MONTHS` and `XRT_TMUNIT_YEARS`.

Time Base

The data's X-values will be measured relative to the time base. For example, if the X-values are days in 1993, (9 means Jan. 10, 31 is Feb. 1 etc.) you will need to set the time base to the beginning of the 1993 year. The `XtNxrtTimeBase` defines the time from which your X-values begin measuring. This resource is a *long* that dynamically

1. This assumes that you haven't fixed the value of `XtNxrtXAxisMin`, `XtNxrtXAxisMax`, `XtNxrtXMin` or `XtNxrtXMax`.

defaults to Jan. 1, 1970 in your time zone. It represents the number of seconds since Jan. 1, 1970 GMT. Since your data is quite likely to begin at a different time, you will almost always have to set this resource when using a time-axis.

The convenience routine *XrtMakeTime()* is useful for converting a normal date and time into the number of seconds since Jan. 1, 1970. The ANSI standard procedure *mktime()* can also be used. For example, if the base time for your data is April 19, 1993, 08:30:05, you could use the following code to set the time base:

```
long tval;
struct tm ltime;

/* Set time base to April 19, 1993, 08:30:05 */
tval = XrtMakeTime(93, 4-1, 19, 8, 30, 5);
XtVaSetValues(graph,
    XtNxrtTimeBase, tval, NULL);
```

Time Format

XRT/graph uses the value of the string specified by `XtNxrtTimeFormat` and your system's ANSI C standard function *strftime()* for formatting the time-axis labels.[1] Since *strftime()* does vary slightly from system to system, you should read your system's documentation to understand the details of how this string is interpreted. A summary is provided with the `XtNxrtTimeFormat` description in Appendix A.

To let XRT/graph calculate an appropriate time format by default, set `XtNxrtTimeFormatUseDefault` to `True`.

The following table shows some examples of how the time 06:55:50 on Monday April 12, 1993 could be formatted:

Time Format String	Example
%Y	1993
%b %y	Apr 93
%b	Apr
%b %d	Apr 12
%a %d	Mon 12
%a	Mon
%a %H:00	Mon 06:00
%H:%M	06:55
%H:%M:%S	06:55:50

To display tick marks at the time annotation, set `XtNxrtXTickUseDefault` to `True`.

1. Since strftime() uses the locale internationalization standards, time-axis annotation can automatically adapt to the various languages and environments supported by locale. Note that you cannot set the timebase to a value less than Jan. 1, 1970 in your time zone, as some systems flag values in 1969 as being invalid.

Time Precision

All data displayed in graphs is represented in single-precision floating-point values. Most architectures use the IEEE standard representation for *float*, which provides 6 digits of accuracy, so it is not possible for XRT/graph to uniquely represent as many time values as ANSI and POSIX (which usually use a 4-byte *long*). The following table indicates the error inherent in a floating-point representation of time values:

Time Period	Number of Seconds	Error
1 Day	86,400	< 1 second
1 Year	31,536,000	< 2 minutes
since 1970	~730 Million	< 17 minutes

This table suggests that, for example, every hour in a year can be accurately represented, but not every minute. To combat this round-off problem, care should be taken in choosing how time units are specified, and `XtNxrtTimeBase` should be assigned the minimum time value that will be encountered.

For instance, suppose you have a year's worth of data, and the X-values are measured in seconds since the beginning of the year. If you set `XtNxrtTimeBase` to the beginning of the year, it will be possible to view the data at any granularity from the whole year down to an hour. If you attempt to zoom in further than this, the values that are represented can be inaccurate by 1-2 minutes. If further zooming is necessary, you should switch to a new dataset representing a single day's data, and set `XtNxrtTimeBase` to the beginning of the day. You can then zoom in to 1 second granularity.

Time-axis Example

The *time.c* program, located in *$XRTHOME/src/graph/examples*, reads in a data file from the file *data/time.dat*. This data file has 12 X-values ranging from 0 to 11, representing the 12 months of the year 1992. When *time.c* is compiled and run, the window below appears:

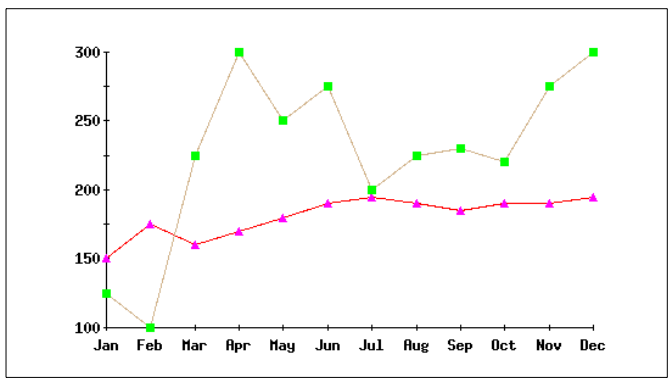


Figure 33 Graph created by the time.c program

```
ARRAY 2 12
# x values, for jan to dec 1992.
0 1 2 3 4 5 6 7 8 9 10 11
# y values line 1
150 175 160 170 180 190 195 190 185 190 190 195
# y values line 2
125 100 225 300 250 275 200 225 230 220 275 300
```

Figure 34 The data/time.dat data file

```
#include <time.h>
#include <Xm/XrtGraph.h>

main(int argc, char **argv)
{
    XtAppContextapp_context;
    Widget      graph, toplevel;
    XrtDataHandle my_data;
    time_t      tval;
    static struct tm ltime;

    /* Allocate and load the data */
    my_data = XrtDataCreateFromFile("data/time.dat", NULL);

    /* Calculate basetime of Jan 1, 1992 */
    ltime.tm_year = 92;
    ltime.tm_mday = 1;
    time.tm_isdst = -1; /* mktime figures out TZ effect */
    tval = mktime(&ltime);

    /* Initialize Intrinsics */
    toplevel = XtVaAppInitialize(&app_context, "Time", NULL, 0, &argc, argv,
             NULL, NULL);
    /* Create graph */
    graph = XtVaCreateManagedWidget("mygraph", xtXrtGraphWidgetClass, toplevel,
            XtNxrtData,              my_data,
            XtNxrtTimeBase,          tval,
            XtNxrtTimeUnit,          XRT_TMUNIT_MONTHS,
            XtNxrtXAnnotationMethod, XRT_ANNO_TIME_LABELS,
            XtNwidth,                500,
            XtNheight,               300,
            0);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app_context);
```

Figure 35 The time.c program

4.3 Axis Controls

This section describes how to program various aspects of the axes, concentrating on the X- and Y-axes. Information on using a second Y-axis (Y2) is found in section 7.1 on page 127.

4.3.1 Axis and Data Bounds

By default, XRT/graph displays all data in the data object. It determines the extent of the axes (`XtNxrt[XYY2]AxisMin` and `XtNxrt[XYY2]AxisMax`) based on the minimum and maximum data values (`XtNxrt[XYY2]Min` and `XtNxrt[XYY2]Max`), the origin, and the numbering increment. You can frame the graph data by either specifying new bounds for the axes or new bounds for the data. Generally, framing the axis gives better results than framing the data, due to the way XRT/graph calculates axis extents.

Axis Min and Max

`XtNxrt[XYY2]AxisMin` and `XtNxrt[XYY2]AxisMax` specify the minimum and maximum axis values to be displayed. Use these resources to frame a graph display precisely at particular *axis* values. For example, the following code "zooms" in on the center of a graph by a factor of 2:

```
float       xmax, xmin, ymax, ymin;
float       xmax2, xmin2, ymax2, ymin2;
XtVaGetValues(graph,
    XtNxrtXAxisMin, &xmin,
    XtNxrtXAxisMax, &xmax,
    XtNxrtYAxisMin, &ymin,
    XtNxrtYAxisMax, &ymax,
    NULL);
xmin2 = xmin + (xmax - xmin) / 4.0;
xmax2 = xmax - (xmax - xmin) / 4.0;
ymin2 = ymin + (ymax - ymin) / 4.0;
ymax2 = ymax - (ymax - ymin) / 4.0;
XtVaSetValues(graph,
    XtNxrtYAxisMin, XrtFloatToArgVal(xmin2),
    XtNxrtXAxisMax, XrtFloatToArgVal(xmax2),
    XtNxrtYAxisMin, XrtFloatToArgVal(ymin2),
    XtNxrtYAxisMax, XrtFloatToArgVal(ymax2),
    NULL);
```

XRT/graph will override any invalid hard-coded axis bounds by setting the corresponding `UseDefault` resource to `True`. For example, if `XtNxrtYAxisMax` is -20 (and `XtNxrtYAxisMaxUseDefault` is `False`) and `XtNxrtYAxisLogarithmic` is set to `True`, `XtNxrtYAxisMaxUseDefault` will be set to `True` because logarithmic axes can only display values greater than zero.

Data Min and Max

`XtNxrt[XYY2]Max` and `XtNxrt[XYY2]Min` specify the minimum and maximum data values to be displayed. Use these resources when you want to frame a graph display at particular *data* values. (User-specified values for these resources are ignored for some graph types.)

4.3.2 Origin

Origin Coordinates

You can specify the coordinates of the origin with `XtNxrt[XYY2]Origin`. For example, to force the Y-axis to cross the X-axis at X=10, set `XtNxrtXOrigin` to 10.0.

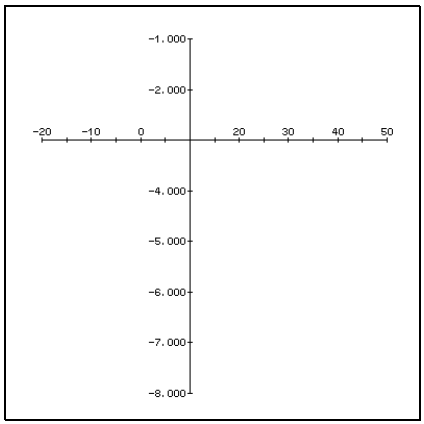


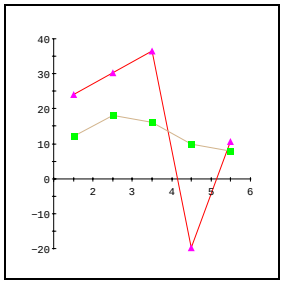
Figure 36 Axes with Origin (10, -3), X Precision is -1, Y Precision is 3

Origin Placement

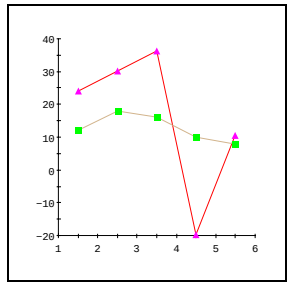
The placement of axis origins can be customized using `XtNxrt[XYY2]OriginPlacement`. Valid values are:

`XRT_ORIGIN_AUTO` (default)	For plot/area graphs, the origin is placed at the minimum axis value or at zero if the dataset contains positive and negative values. For bar, stacking bar, HiLo, HiLoOpenClose and candle graphs, the origin is placed at zero.
`XRT_ORIGIN_ZERO`	Origin is placed at 0.
`XRT_ORIGIN_MIN`	Origin is placed at the minimum axis value.
`XRT_ORIGIN_MAX`	Origin is placed at the maximum axis value.

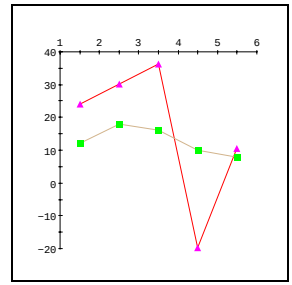
This resource is ignored when `XtNxrt[XYY2]OriginUseDefault` is `False`; setting an origin explicitly overrides this resource.



YOriginPlacement = _ORIGIN_AUTO



YOriginPlacement = _ORIGIN_MIN



YOriginPlacement = _ORIGIN_MAX

Figure 37 Effect of Origin Placements

4.3.3 Tick and Grid Increments

Tick Increment

The increment between tick marks drawn along an axis can be controlled using `XtNxrt[XYY2]Tick`. These resources are similar to `XtNxrt[XYY2]Num`. Numbering increments do not have to be multiples of tick increments.

Gridlines

`XtNxrt[XYY2]Grid` specifies the spacing between gridlines. By default, these resources are 0.0, resulting in no gridlines at all. To display gridlines at each axis label, set `XtNxrt[XYY2]GridUseDefault` to `True`. Alternately, you can specify your own custom increment by setting `XtNxrt[XYY2]Grid`.

The style of the grid is determined by `XtNxrt[XYY2]GridDataStyle` and `XtNxrt[XYY2]GridDataStyleUseDefault`.

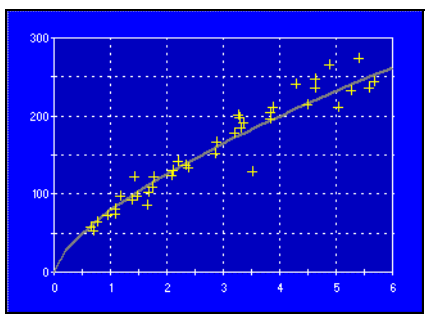


Figure 38 Gridlines and axis tick marks

4.3.4 Axis Title and Annotation Rotation

Title

The `XtNxrt[XYY2]Title` resources may be used to specify a title for each axis. These resources are not used in polar, radar, or filled radar charts.

Title Rotation

The title on the vertical axes (that is, the Y- and Y2-axis unless `XtNxrtInvertOrientation` is `True`) can be rotated 90 degrees counter-clockwise by setting `XtNxrt[XYY2]TitleRotation` to `XRT_ROTATE_90`.

You can also rotate titles 270 degrees counter-clockwise by setting the rotation resource to `XRT_ROTATE_270`. 270-degree rotation usually looks best on the Y2-axis. To return to horizontal titles, set the rotation resource to `XRT_ROTATE_NONE`.

Titles on the horizontal axis (that is, the X-axis unless `XtNxrtInvertOrientation` is `True`) cannot be rotated.

Annotation Rotation

Axis annotation can be rotated using `XtNxrt[XYY2]AnnotationRotation`. X-axis annotation rotation is ignored on polar, radar, and filled radar charts.

Rendering of rotated text can slow down your real-time performance dramatically (a factor of 10 is not uncommon). Do not use rotated text if real-time performance is critical to your application.

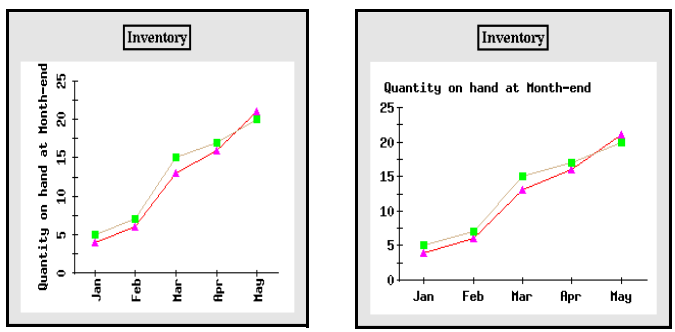


Figure 39 Rotating Annotation and Title by 90 Degrees

4.3.5 Other Axis Controls

Orientation

When `XtNxrtInvertOrientation` is `False`, all the X-axis resources apply to the horizontal axis, and the Y-axis resources apply to the vertical axis. When `XtNxrtInvertOrientation` is `True`, the X-axis is vertical, and the Y-axis is horizontal.

Margins

You can adjust the space between the axes and the edge of the graph area with the margin resources (`XtNxrtGraphMarginTop`, `XtNxrtGraphMarginBottom`, `XtNxrtGraphMarginLeft` and `XtNxrtGraphMarginRight`).[1] Each resource specifies the number of pixels between one edge of the graph area and its axis. By default, XRT/graph calculates margins automatically (the `UseDefault` resources are `True`), setting enough space to display axis annotation and titles.

Margins can be used to:

- Line up the axes of several graph widgets; for example, use `XtNxrtGraphMarginLeft` to line up several graphs along the Y-axis.
- Scale a graph; for example, "zoom in" by setting all the Margin resources to negative values, "zoom out" by setting them to positive values.
- Translate a graph; for example, increase `XtNxrtGraphMarginTop` and decrease `XtNxrtGraphMarginBottom` by the same amount to shift the graph down in the graph area.

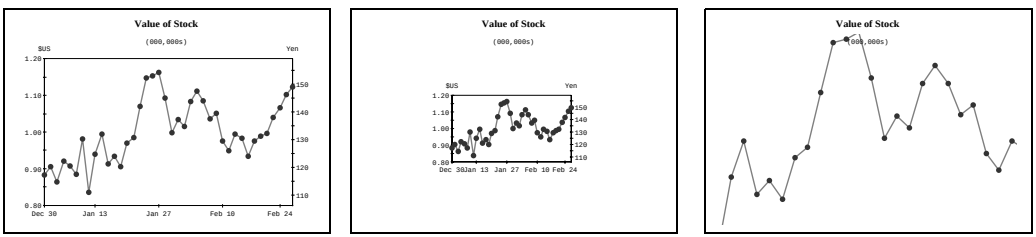


Figure 40 Using Margins to scale a graph within the graph area

Axes Font

By default, the font used for axes numbering and titles is the "fixed" font. You may change the font to any font available on your Server using the `XtNxrtAxisFont` resource. Changing the axes font is similar to changing the header, footer or legend font, as discussed in section 4.8 on page 72.

Reversed Axis

You can change the direction of the X-, Y-, and Y2-axes by setting `XtNxrt[XYY2]AxisReversed` to `True`.

Once a relationship has been specified between the Y- and Y2- axis, you can only specify the direction of the Y-axis. If the relationship multiplier is negative, the direction of the Y2-axis will be the opposite of the Y-axis.

1. Margins can be adjusted interactively by an end-user. See section 6.1 on page 109 for more information on interactively scaling, translating, and zooming graphs.

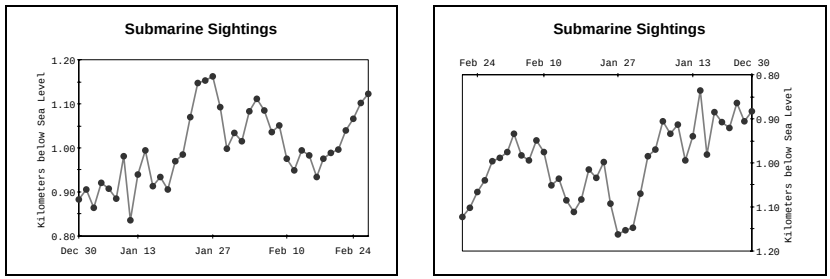


Figure 41 Normal and Reversed X- and Y-axes

Annotation/Title Placement

You can control the placement of axis annotation and titles with `XtNxrt[XYY2]AnnoPlacement`. You can fix each axis' annotation/title at the origin, the axis minimum, or the axis maximum. Valid values are:

`XRT_ANNO_AUTO` (default)	Annotation is placed automatically. For plot and area graphs, annotation is placed at the origin. For other graph types, annotation is placed at the end of the axis closest to the origin.
`XRT_ANNO_ORIGIN`	Annotation is placed at the origin.
`XRT_ANNO_MIN`	Annotation is placed at the minimum axis value.
`XRT_ANNO_MAX`	Annotation is placed at the maximum axis value.

Y-axis annotation placement is ignored on dual-axis graphs and Y2-axis placement is ignored unless Y2 is the primary Y-axis.

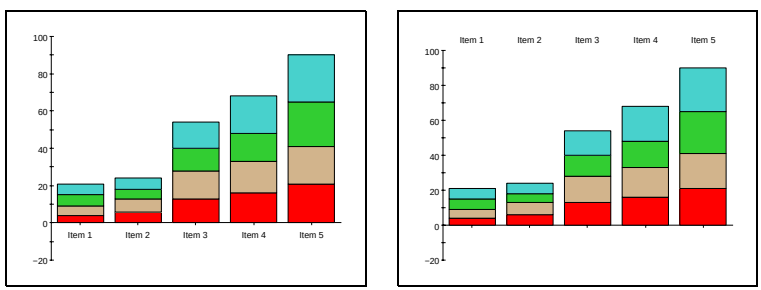


Figure 42 Placing X-Axis Annotation at Y-Origin and Y-Maximum

Axis Show

`XtNxrt[XYY2]AxisShow` specifies whether XRT/graph should draw the axis at all. If `False`, the axis will not be drawn.

Bounding Box

To cause a box to be drawn surrounding the data area, set `XtNxrtAxisBoundingBox` to `True`.

Axis Data Styles

How an axis looks when it is displayed (color, width and pattern) is controlled by the `XtNxrt[XYY2]AxisDataStyle`.

For more information on this and other data styles, see section 4.9 on page 74.

4.4 Annotation Formatting

When the annotation method[1] of an axis is Values, you can display the numeric annotation in a variety of custom numeric, accounting, currency, percentage or scientific formats. Use the `XtNxrt[XYY2]AxisLabelFormat` resource to define the axis' formatting specification, a string containing formatting codes to apply to each axis label. Figure 43 shows several different format specifications. The following code sets the Y-axis format to the value of `my_format`:

```
char * my_format = "$#,##0.00;-$#,##0.00";
XtVaSetValues(graph,
    XtNxrtYAxisLabelFormat, my_format,
    NULL);
```

1. Specified by the `XtNxrt[XYY2]AnnotationMethod` resource.

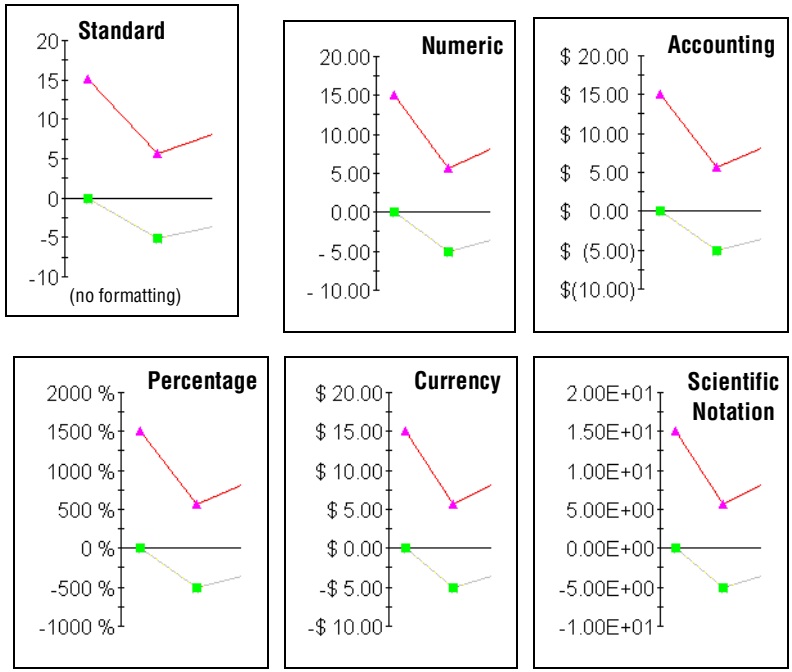


Figure 43 Effect of different axis format specifications

Format Commands

A format specification contains one or more format commands, such as "#", "0", "$", ".", or "[multiplier 0.1]". You can define different groups of format commands for positive, zero, and negative values by separating each group with semicolons (";"). The *first* group always covers positive numbers; it includes zero if two or less groups are defined, and negative numbers if there is only one group. The *second* group is always used for negative numbers, and the *third* is always used for zero. You can define more groups for use with special ranges of numbers; this is discussed later in this section.

Common Format Specifications

The following table lists format specifications for common types of formatting:

	Format Specification	Examples of Formatted Axis Numbering			
No formatting	`NULL`	-20	0	140	1200
Number Number of digits, rounding	`0.00;-0.00` `#,##0.0`	-20.00 -20.0	0.00 0.0	140.00 140.0	1200.00 1,200.0
Currency	`$#,##0.00;-$#,##0.00` `$#,##0.00;$(#,##0.00)`	-$20.00 $(20.00)	$0.00 $0.00	$140.00 $140.00	$1,200.00 $1,200.00

	Format Specification	Examples of Formatted Axis Numbering
Accounting Aligns decimal and currency symbol	`_-$* #,##0.00;-$* #,##0.00` `$* _(#,##0.00_);$* (#,##0.00)`	-$ 20.00 $ 0.00 $ 140.00 $1,200.00 $ (20.00) $ 0.00 $ 140.00 $1,200.00

Percentage
Multiplies by 100, adds % sign
	`0.00%;-0.00%`
	`0 %;-0 %`
	`%0;%-0`
		-2000.00% 0.00% 14000.00%
		-2000 % 0 % 14000 %
		%-2000 %0 %14000

Fraction	`0 ?/?` `0 ?/8`	-5 1/4 0 140 3/5 -5 2/8 0 1200 7/8

Scientific	`0.00E+00`
	`0.00e-00`
	`0.0[hide]E[show]+00`
		-2.00E+01 0.00E+00 4.00E+01
		-2.00e01 0.00e00 4.00e01
		-2.0+01 0.0+00 4.0+01

Format Codes

The following lists and describes the valid format codes you can use in a format specification.

`# 0 ?` — Creates a placeholder for a number. Use with `.` (period) code to specify the number of digits to display. `#` does not display extra zeros. `0` displays extra zeros when there are fewer digits than in the format. `?` works like `0` and adds space for insignificant zeros to align decimal points and fractions.
For example, "#.0#" formats 16 as 16.0, and 98.666 as 98.67.

`.` (period) — Positions the decimal point, used with `#`, `0`, and `?` to specify the number of digits and rounding.
For example, "###.0000" formats 123.45 as 123.4500.

`,` (comma) — When between digits, acts as a thousands separator. When at the end of a value, scales value by a thousand.
For example, "#,###" formats 21000 as 21,000.
"0.0," formats 1200 as 1.2.

`$ - + / ( )` — Inserts character where it appears in the format.
For example, "$#.00" formats 23.3 as $21.30.

`\\<character>` — Inserts `<character>` where it appears in the format.
For example, "\\%" inserts a percent sign (without multiplying).

`_<character>` — Inserts a space the width of `<character>`.
For example, "_-" displays a space the width of the minus sign.

`%` — Multiplies value by 100 and inserts % character.
For example, "#.##%" formats .85 as 85%.

`[multiplier <n>]` — Multiplies following value by `<n>`.
For example, "[multiplier 1.6]" formats 10 as 16.

`E+ e+ E- e-`	Converts value to scientific notation, placing "E" or "e" in front of the exponent. `E+` and `e+` displays "+" before positive exponents and "-" before negative exponents. `E-` and `e-` only displays "-" before negative exponents. For example, "0.0E+00" formats 18987 as 1.8E+04.
`*<character>`	Pads the annotation space with `<character>`. For example, "$* ##.##" displays as much space as possible between the $ and value.
`[hide] [show]`	`[hide]` hides the value or formats that follow. `[show]` turns off format/value hiding. For example, "0.0[hide]E[show]+00" formats 70 as 7.0+01.
; (semicolon)	Ends one group and begins another format group. For example, "`###.##;-###.##;#.#;`".

Defining Special Ranges for Formatting

Several format codes enable you to define formatting for a range of numbers, in finer details than the built-in positive, negative and zero groups. The codes that define conditional formatting are `[>value]`, `[<value]`, `[=value]`, `[<>value]`, `[>=value]`, and `[<=value]`. Use these codes to define new groups whose format codes are applied only when the condition is true. Each group is separated by a semicolon (;). The following example defines special formatting for values greater than 1000:

```
###.##;-###.##;#.#;[>1000]0.0E+00
```

Building Format Specifications

Determining the right combination of format codes can be tricky and time-consuming. To validate a format before setting it on `XtNxrt[XYY2]AxisLabelFormat`, call *XrtCustomFormatValidate()*. This procedure is documented in Appendix D on page 211.

4.5 Positioning Graph Areas

Each of the four areas (header, footer, legend, and graph) will by default be positioned by XRT/graph at runtime. The default positioning for each area depends on a large number of factors, including:

- The window's current width and height.
- The size of the legend, header, and footer areas. These, in turn, depend on the text and fonts being used.
- The value of the `XtNxrtLegendAnchor` resource.
- The positioning of areas which have been explicitly positioned by the user program.

XRT/graph's default positioning algorithms will size and position the header, footer and legend areas first. The graph area will be sized and positioned to fit into the largest remaining rectangular area.

A program can determine and adjust area positioning through the use of the positioning resources.

```
XtNxrtHeaderXXtNxrtHeaderY(XtNxrtHeaderWidth)(XtNxrtHeaderHeight)
XtNxrtFooterXXtNxrtFooterY(XtNxrtFooterWidth)(XtNxrtFooterHeight)
XtNxrtLegendXXtNxrtLegendY(XtNxrtLegendWidth)(XtNxrtLegendHeight)
XtNxrtGraphXXtNxrtGraphYXtNxrtGraphWidthXtNxrtGraphHeight
(Resources in Parentheses are Read-Only)
```

Figure 44 Area Positioning Resources

When used in an *XtGetValues()* call, positioning resources will return the values used the last time the graph was displayed. These resources do not have meaningful values until they are either explicitly set, or a graph has been displayed at least once.

A program should not explicitly set any of the positioning resources unless it is prepared to recalculate them when the window's size changes. See section 6.7 on page 125 for more information on handling window resizing.

In some situations, it may be worthwhile to explicitly set some or all of the positioning resources. For example, if the program will be displaying data that changes in real-time, the default positioning may change slightly with each redisplay. These small positioning changes could distract the user. In this situation, the program should use one of the following strategies:

- Hardcode all the positioning resources. Window resizing should not be allowed.

or

- Let XRT/graph calculate default positioning for all areas when the graph first displays, and for the first display after any window resize. After the first display, explicitly set the positioning resources to the values calculated by XRT/graph.

4.6 3D Effect

Data in bar, stacking bar and pie charts can be displayed with a three-dimensional appearance by setting the `XtNxrtGraphDepth`, `XtNxrtGraphInclination` and `XtNxrtGraphRotation` resources.[1] These resources have no effect on any other graph type or on combination graphs.

Graph depth is the apparent depth as a percentage of graph width. Inclination is the eye's position above the X-axis, measured in degrees. Rotation is the number of

1. If `XtNxrtGraphDepth` is greater than zero, rotation and inclination (inclination only on pie graphs) can be adjusted interactively by an end-user. See section 6.1 on page 109 for more information.

degrees the eye is positioned to the right of the Y-axis. Rotation has no effect on pie graphs.

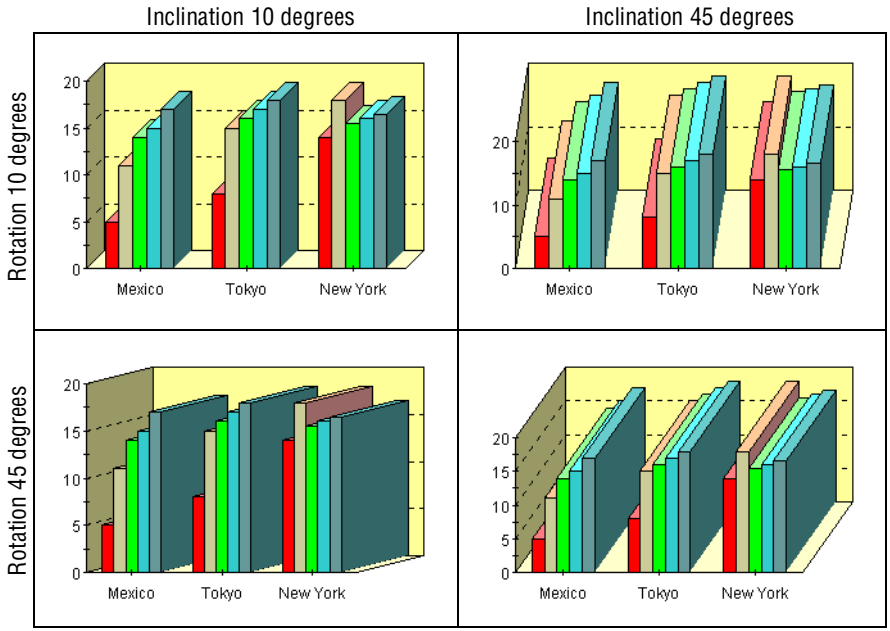


Figure 45 3D Effect on Bar Graphs with Depth equal to 30

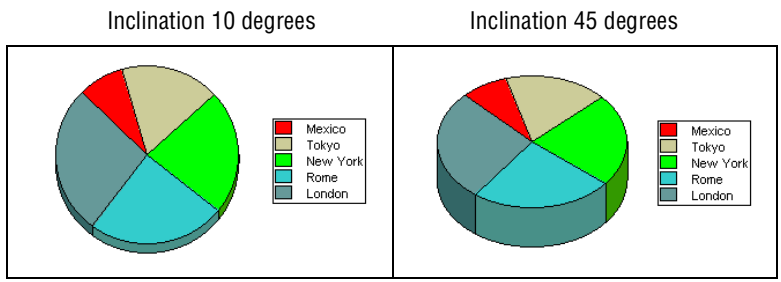


Figure 46 3D Effect on Pie Graphs with Depth equal to 30

Shading Method

XRT/graph by default allocates two additional colors for the 3D shading of each bar or pie slice; a darker shade, and a lighter one. You can reduce the number of colors used by the chart using the `XtNxrtGraph3dShading` resource. When set to `XRT_SHADING_DITHERED`, XRT/graph simulates the light/dark shading without using additional colors by dithering the bar or pie color. When set to `XRT_SHADING_NONE`, XRT/graph simply uses the bar or pie color itself for the shading.

4.7 Stacking

Data in plot, area, polar, radar and filled radar graphs can be displayed in stacking mode when the data type is an array by setting `XtNxrtIsStacked` to `True`. `XtNxrtIsStacked2` applies to second dataset in a combination graph. Excluded sets/series will not be part of the stack. As well, hidden sets are treated as if they were excluded and are thus not part of the stack. **Note:** Negative values can produce some overlapping of points.

100% Axis Fill

Stacked graphs can have their Y-axes expressed as a percentage of the graph's data using the `XtNxrt[YY2]Axis100Percent` resources. These properties are only available when the source data type is an array.

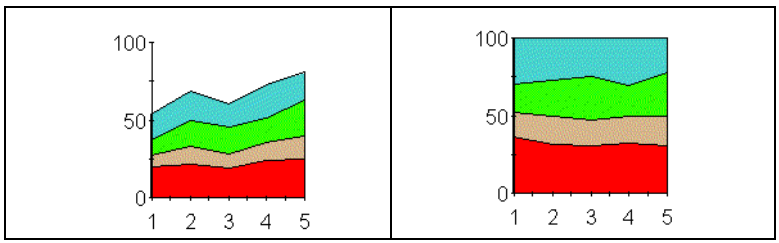


Figure 47 Stacked/area graph using regular and 100-percent Y-axis

4.8 Header, Footer and Legend Display

Unless a program has explicitly positioned areas, XRT/graph will always attempt to display the header at the top, and the footer at the bottom of the window. The default legend position depends on the value of the `XtNxrtLegendAnchor` resource. See section 4.5 on page 69 for details on overriding the default area positioning.

Header & Footer Text

The header and footer areas can both contain multiple lines of text. The text will be aligned within the area, depending on the value of the `XtNxrtHeaderAdjust` or `XtNxrtFooterAdjust` resource. The values `XRT_ADJUST_LEFT`, `XRT_ADJUST_RIGHT` and `XRT_ADJUST_CENTER` cause the text to be left-justified, right-justified or centered. `XRT_ADJUST_CENTER` is the default.

The text for the header and footer areas is specified using the `XtNxrtHeaderStrings` and `XtNxrtFooterStrings` resources. Both of these resources have `NULL`-terminated arrays of strings as their values.

The code below will set two header lines and left-adjust them:

```
static char *hs[] = { "Experiment A Results",
            "Last 60 Days", NULL };
XtVaSetValues(graph,
    XtNxrtHeaderStrings,hs,
    XtNxrtHeaderAdjust, XRT_ADJUST_LEFT,
    NULL);
```

As described in section 4.17 on page 91, the header, footer, legend, graph and data areas may be enhanced with a background image. As well, data (text) areas, headers and footers may use the images themselves as labels, obviating the need for any text. But, this does not apply to all images. Areas where there's no text-based minimum size (graph areas, plot areas, widget backgrounds, etc.) or in the legend do not apply.

It's important to note that though text in these labels can be rotated, rotation does NOT affect the image.

Legend

The text displayed in the legend corresponds to the Set-labels. For more information on Set-labels, see section 4.2.2 on page 52.

By default, XRT/graph will attempt to list the legend contents vertically, and position the legend to the right of the graph area.

The legend layout is controlled through the `XtNxrtLegendOrientation` resource. It may be either `XRT_ALIGN_VERTICAL` or `XRT_ALIGN_HORIZONTAL`. If the legend is too large to fit in one row or column, XRT/graph will attempt to layout the legend in several rows or columns.

The default legend positioning relative to the graph area is controlled with the `XtNxrtLegendAnchor` resource. Valid values correspond to the eight points of the compass: `XRT_ANCHOR_NORTH`, `XRT_ANCHOR_SOUTH`, `XRT_ANCHOR_EAST`, `XRT_ANCHOR_WEST`, `XRT_ANCHOR_NORTHWEST`, `XRT_ANCHOR_NORTHEAST`, `XRT_ANCHOR_SOUTHEAST`, and `XRT_ANCHOR_SOUTHWEST`.

The `XtNxrtLegendReversed` resource allows the set entries to appear in reverse order. A useful application of this resource is for stacked charts, as reversing the legend will display the values in the same top-to-bottom order as the displayed stacks.

The `XtNxrtLegendTitle` resource allows the user to define a legend title. A single line is allowed and it will be centered above the legend. The title uses the legend font.

HiLo, HiLoOpenClose, Candle and Bubble Chart Legends

A legend entry is drawn for each logical set that has a set label. The label for a logical set is based on the logical index of the set, not the physical indices. For each entry in the legend, a line and a point symbol are drawn using the characteristics of the logical set. The size of the point drawn is based on the legend font size, but is no larger than the maximum point symbol size.

Font Specification

A font may be specified for each of the header, footer and legend areas and also for the axes annotation. XRT/graph can use any font supported by the X Server. For information on International font support, see section 7.5 on page 133.

If you choose to use a named font, use the X Window program *xlsfonts*, or the Xlib call *XListFonts()* to determine which fonts are available on the Server. Use the Xt

procedure *XtConvertAndStore()* to obtain the XmFontList from a font string. See your Xt programming documentation for details on finding and setting fonts.[1]

```
static XmFontList
CvtStringToFontList(parent, font_name)
Widget  parent;
char    *font_name;
{
    XrmValue    to;
    XrmValue    from;

    from.addr = font_name;
    from.size = strlen(font_name);
    to.addr = NULL;
    to.size = 0;

    if (!XtConvertAndStore(parent, XmRString, &from, XmRFontList, &to)) {
        fprintf(stderr, "Conversion of %s to XmFontList failed!\n", font_name);
        return((XmFontList)0);
    }
    return(*(XmFontList *)to.addr);
}


XmFontList hf, ff;
hf = CvtStringToFontList(graph, "-*-times-medium-*-*--18-*-iso8859-*");
ff = CvtStringToFontList(graph, "9x15");
if (hf != NULL && ff != NULL) {
    XtVaSetValues(graph,
                XtNxrtHeaderFont,  hf,
                XtNxrtFooterFont,  ff,
                NULL);
}
```

Figure 48 Setting XmFontList resources

4.9 Data Styles

How a data value looks when it is displayed (i.e. color, line pattern, fill pattern, point style, line thickness etc.) depends on the data style that has been defined for that data value. For example, the values in the third set of data will be rendered on screen using the third data style.

Default Data Styles

XRT/graph provides default data styles for monochrome and color Servers. If `XtNxrtDataStylesUseDefault` is `True`, the graph will use the appropriate (color or mono) default data styles. Even after setting explicit data styles, a program may again use default data styles by setting `XtNxrtDataStylesUseDefault` to `True`.

1. XRT/graph's Property Editor can be used to examine and choose fonts.

The XrtDataStyle structure is declared in *<Xm/xrtgfcom.h>*:

```
typedef struct {
    XrtLinePattern  lpat;       /* line pattern */
    XrtFillPattern  fpat;       /* fill pattern */
    char *          color;      /* color */
    int             width;      /* line width */
    XrtPoint        point;      /* point style */
    char *          pcolor;     /* point color */
    int             psize;      /* point size */
    unsigned long   res1;       /* reserved */
    unsigned long   res2;       /* reserved */
    unsigned long   resh;       /* reserved */
    unsigned long   ress;       /* reserved */
} XrtDataStyle;
```

The XrtDataStyle data structure contains all the information about how a set of data will be represented graphically. The fields are broken down as follows:

lpat	The line pattern used for plots as defined in *<Xm/xrtgfcom.h>*. Must be one of the XRT_LPAT_ constants listed in Figure 49.
fpat	The fill pattern used in area graphs and bar and pie graphs, as defined in *<Xm/xrtgfcom.h>*. Must be one of the XRT_FPAT_ constants listed in Figure 50.
color	The color used when drawing lines in plots and for fills in area graphs and bar and pie graphs. It is either a named color or a '#' character followed by two hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.
width	The line width used for plots. Must be greater than or equal to 1.
point	The point style used for plots, as defined in *<Xm/xrtgfcom.h>*. Must be one of the XRT_POINT_ constants listed in Figure 51.
pcolor	The point color used for points in plots. It is either a named color or a '#' character followed by two hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.
psize	The size of points that appear in plots. Must be greater than or equal to 0. A size of 0 will result in no point being drawn. A point size is a relative measure. Do not assume that a point size of 12 means that the point's glyph will be exactly 12 pixels from top to bottom.

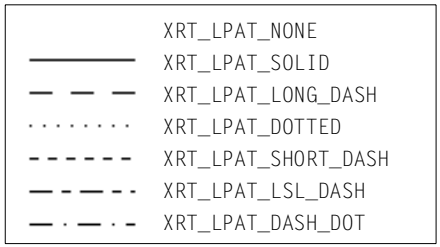


Figure 49 Line Patterns

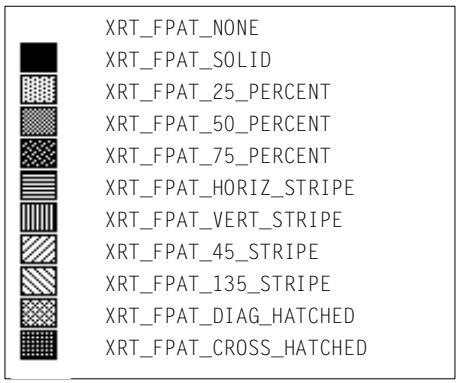


Figure 50 Fill Patterns

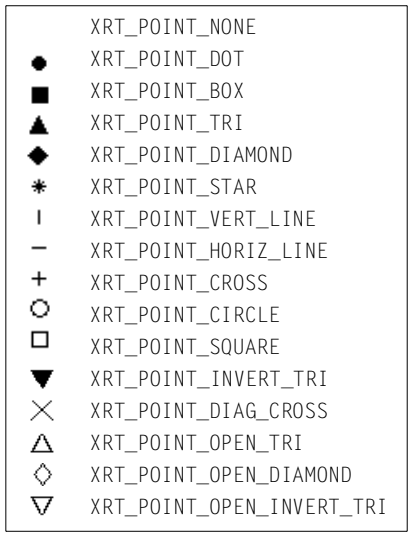


Figure 51 Point Styles

Explicitly Setting Data Styles

It is usually satisfactory to use XRT/graph's default data styles. If a program explicitly sets any data styles, it is responsible for ensuring that the data styles are appropriate to the type of display being used. For example, a solid red bar and a solid green bar will both appear the same when rendered on a monochrome display.

A side effect of setting any data style element explicitly is that the `XtNxrtDataStylesUseDefault` resource will be set to `False`. XRT/graph will use the default data styles for any sets of data without an explicit data style. The most straightforward way to set data styles in a program is shown in Figure 52.

```
static XrtDataStyle color_styles[] = {
/* 0 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "red", 1,  XRT_POINT_TRI,
        "magenta", XRT_DEFAULT_POINT_SIZE },
/* 1 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "tan", 1,  XRT_POINT_BOX,
        "green", XRT_DEFAULT_POINT_SIZE },
/* 2 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "LimeGreen", 1,  XRT_POINT_DOT,
        "medium blue", XRT_DEFAULT_POINT_SIZE },
/* 3 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "MediumTurquoise", 1,
        XRT_POINT_DIAMOND, "red", XRT_DEFAULT_POINT_SIZE },
/* 4 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID,    "blue", 1,   XRT_POINT_STAR,
        "red", XRT_DEFAULT_POINT_SIZE },
/* 5 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "magenta", 1,   XRT_POINT_TRI,
        "medium blue", XRT_DEFAULT_POINT_SIZE },
/* 6 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "green yellow", 1,   XRT_POINT_BOX,
        "magenta", XRT_DEFAULT_POINT_SIZE },
/* 7 */ { XRT_LPAT_SOLID, XRT_FPAT_SOLID, "medium blue", 1,   XRT_POINT_DOT,
        "light sky blue", XRT_DEFAULT_POINT_SIZE },
};

static XrtDataStyle *my_dstyles_ptr[] = {
    &color_styles[0],
    &color_styles[1],
    &color_styles[2],
    &color_styles[3],
    &color_styles[4],
    &color_styles[5],
    &color_styles[6],
    &color_styles[7],
    NULL
};

...
    Widget  graph;
...
    graph = XtVaCreateManagedWidget("my_graph",
        xtXrtGraphWidgetClass, parent,
...
        XtNxrtDataStyles,      my_dstyles_ptr,
        NULL );
```

Figure 52 Setting Data Styles

Accessing Data Styles

Data styles are accessed through the `XtNxrtDataStyles` resource and/or through the *XrtSetNthDataStyle()* and *XrtGetNthDataStyle()* methods.

`XtNxrtDataStyles` can be used to get or set the entire array of data styles. When used with *XtGetValues()*, the returned array pointer should be considered read-only. The *XrtDupDataStyles()* convenience routine may be used to duplicate the returned

data styles array pointer. For example, to set all line widths to 5-pixels thick, the following code could be used:

```
int             i;
XrtDataStyle    **ds, **myds;
XtVaGetValues(graph, XtNxrtDataStyles, &ds, NULL);
myds = XrtDupDataStyles(ds);
for(i=0; myds[i]; i++){
    myds[i]->width = 5;
}
XtVaSetValues(graph,
    XtNxrtDataStyles, myds,
    NULL);
XrtFreeDataStyles(myds);
```

A program can get a pointer to a particular `XrtDataStyle` structure using *XrtGetNthDataStyle()*. For example, if `graph` is a plot of the population data in Figure 30, the following code will print out the line thickness being used for the set of Tokyo data, and double it:

```
XrtDataStyle *dstyle, my_dstyle;
dstyle = XrtGetNthDataStyle(graph, 1);
printf("line width for 2nd set of data: %d\n", dstyle->width);
my_dstyle = * dstyle;
my_dstyle.width = my_dstyle.width * 2;
XrtSetNthDataStyle(graph, 1, &my_dstyle);
```

Axis Data Styles

Individual axes can be displayed with different color, width and line patterns using `XtNxrt[XYY2]AxisDataStyle` and `XtNxrt[XYY2]AxisDataStyleUseDefault`.

If there are two Y axes and Y2 is subordinate, then Y2 will use the axis data style of the Y-axis. Reciprocally, if Y2 is primary, then Y will use its axis data style. In either case, only the main axis line (the one with the ticks) will be affected.

PointStyle Data Styles

PointStyles are used to alter the behavior of a specific point of data, such as the datastyle of an individual data point or to set special options for pie slices, for example exploding pies. PointStyle objects are referenced by a handle of type `XrtPointStyleHandle`. The following is a summary of the routines used to assign behavior to specific points. Complete reference information is in Appendix C on page 211.

PointStyle Object Summary	
XrtPointStyleCreate()	Creates a PointStyle object as a child of the chart.
XrtPointStyleDestroy()	Destroys a PointStyle object.
XrtPointStyleSetValues()	Sets the current value of one or more PointStyle object attributes.
XrtPointStyleGetValues()	Retrieves the current value of one or more PointStyle object attributes.

Note: A pointstyle must be positioned by `XRT_POINTSTYLE_SET` and `XRT_POINTSTYLE_POINT` before it can be used.

Interactive Data Style Editing

The XRT/graph Property Editor enables you to interactively edit data styles. The *feedback* and *vmgraph* demos contain their own data style editor – the source code is found in *$XRTHOME/src/graph/tools/mtgraph.c*.

4.10 Special Bar Chart Resources

Bar charts have several resources that control exactly how the bars are spaced and displayed. Bar charts have their bars grouped into *clusters*. There is a cluster for each point.

Stacking Bar Charts

Data is displayed as a stacking bar chart when `XtNxrtType` is set to `XRT_TYPE_STACKING_BAR`. In stacking bar charts, there is only one bar per cluster. All Y-values less than zero are stacked below the Y-origin (normally the X-axis).

Cluster Width

In non-stacking bar charts, bars are grouped into clusters. To specify how much of the available cluster space should be occupied by the bars, use the `XtNxrtBarClusterWidth` resource. By default, `XtNxrtBarClusterWidth` is 50, which means that the bars will occupy 50% of the available cluster space.

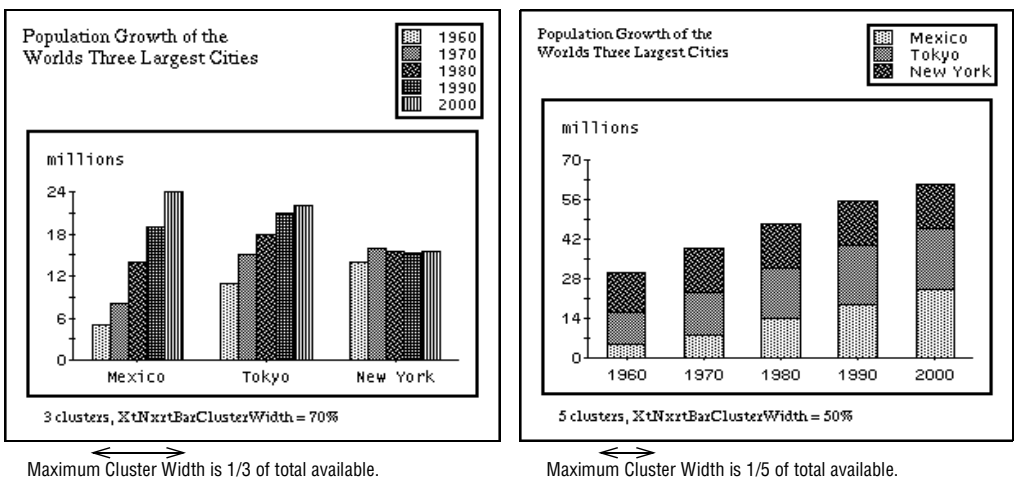


Figure 53 Cluster Width in Bar Charts and Stacking Bar Charts

Cluster Overlap

The spacing between bars in the same cluster is controlled using the `XtNxrtBarClusterOverlap` resource. Cluster overlap is specified as a percentage of the bar's width. This resource has no effect on stacking bar charts because they have only one bar per cluster.

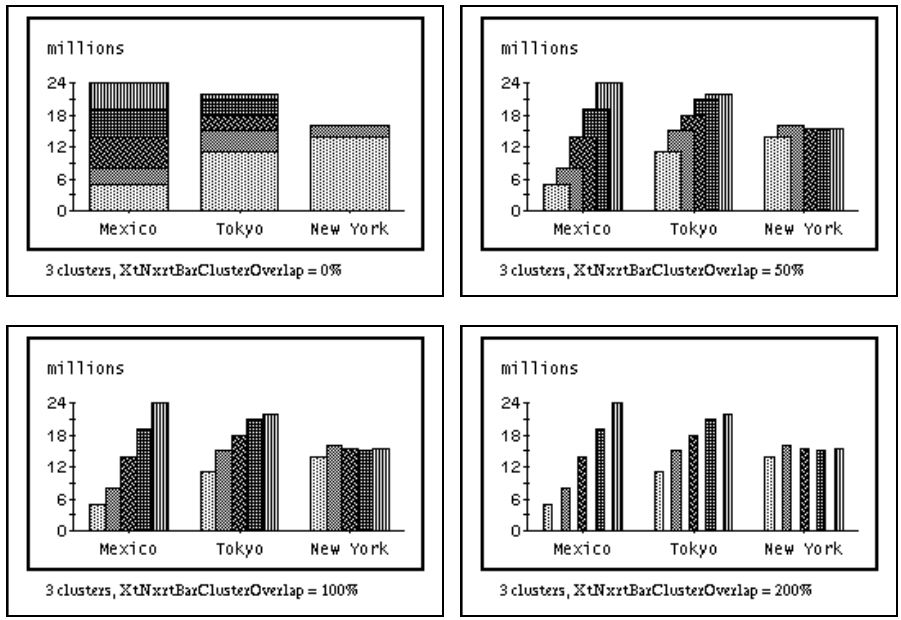


Figure 54 Cluster Overlap

100-Percent Stacking Bar Charts

The Y-axes of stacking bar charts can switch to a percentage interpretation of the bar data using the `XtNxrt[YY2]Axis100Percent` resources. When set to `True`, each stacked bar's total Y-values represents 100%. The Y-value of each bar is interpreted as its percentage of the total. These resources have no effect on bar charts.

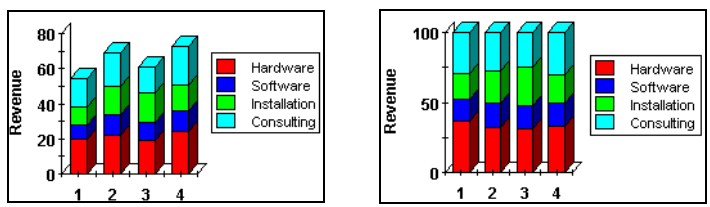


Figure 55 Stacking Bar Chart using regular and 100-percent Y-axis

4.11 Special Pie Chart Resources

Pie charts are significantly different from other types of graphs, such as plot and bar charts, since they are not based on a 2-dimensional grid. Pie charts also introduce the concept of an *other* category, into which all values below a specified threshold are grouped.

XRT/graph will draw a pie chart comparing each point across all sets. To begin, the sum of all of the first points is calculated (then all of the second points, etc.). Then the percentage of the sum is calculated for each point. It is suggested that data objects with large numbers of points not be displayed as pie charts. Negative data values are treated as zero when displayed in a pie chart.

To compare the relative sizes of each of the points within a set, transpose the dataset using *XrtDataTranspose()*. For more information, see section 5.5 on page 104.

Because of the unique nature of pie charts, XRT/graph defines a few resources which pertain only to graphs when `XtNxrtType` is `XRT_TYPE_PIE`.

Other Slice Threshold

There are two factors which contribute to threshold interpretation. *Threshold method* and *threshold value* determine which values will be displayed with their own slice, and which will contribute to the *other* slice.

If `XtNxrtPieThresholdMethod` is `XRT_PIE_SLICE_CUTOFF`, XRT/graph will examine the value of `XtNxrtPieThresholdValue`. All data values whose percentage of the total is less than the *threshold value* are grouped into the *other* slice.

If `XtNxrtPieThresholdMethod` is `XRT_PIE_PERCENTILE`, as many of the smallest data values as necessary are grouped into the *other* slice so that the *other* slice is less than or equal to the *threshold value* percent of the total.

The `XtNxrtPieThresholdValue` defines a floating-point value between 0.0 and 100.0. To disable creation of the *other* slice, set `XtNxrtPieThresholdValue` to 0.0.

You can determine which sets were grouped into the other slice for a point using the *XrtGetOtherSliceSets()* method, described in Appendix D on page 211.

An example of various threshold methods is shown in Figure 56.

Other Slice Label

The label used to annotate the *other* slice is determined by `XtNxrtOtherLabel`. By default, the *other* slice is labelled “Other”. This resource can be used to specify a more appropriate label for international locales.

Other Slice Data Style

`XtNxrtOtherDataStyle` specifies the data style to be used for displaying the *other* pie slice. See section 4.9 on page 74 for more information on the `XrtDataStyle` structure. By default, XRT/graph will choose a unique data style for the *other* slice. An application can go back to using the default data style for the *other* slice by setting `XtNxrtOtherDataStyleUseDefault` to `True`.

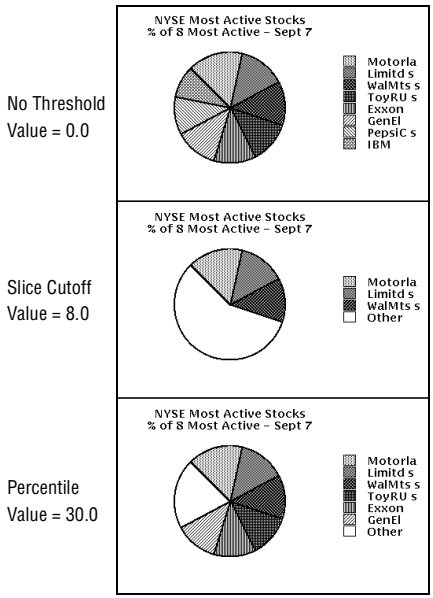


Figure 56 Threshold Methods

Pie Ordering

The order of pie slices around the pie is determined by `XtNxrtPieOrder`. It must be one of `XRT_PIEORDER_ASCENDING`, `XRT_PIEORDER_DESCENDING` or `XRT_PIEORDER_DATA_ORDER`. If it is `XRT_PIEORDER_DATA_ORDER`, the slices not contributing to *other* are displayed in the same order that they appear in the data. The *other* slice is always the last slice in any ordering.

Minimum Slices

`XtNxrtPieMinSlices` specifies a minimum number of slices to display. This resource has precedence over the threshold method and value. If, for example, `XtNxrtPieMinSlices` is set to 5, XRT/graph will always try to display 5 slices before grouping any data into the *other* slice.

All-Zero Pies

If all of a pie's data values are zero (or negative), no pie will be drawn and it will not be possible to set a marker on the pie. Also, the *XrtPick()* procedure will indicate the correct pie index (as if the pie had been drawn), but the slice index will always be zero. *XrtPick()* will always return a non-zero distance value for all-zero pies.

Exploding Pies

A member of a pie's data set (a slice) can be separated from the main pie figure, or *exploded,* by using `XRT_POINTSTYLE_SLICE_OFFSET`, a member of the PointStyle datastyle. This offsets the slice from the center of the pie. The offset is expressed as a percentage of the pie's radius.

For more information on the offset resource and its associated PointStyle resources, see PointStyle Data Styles on page 79.

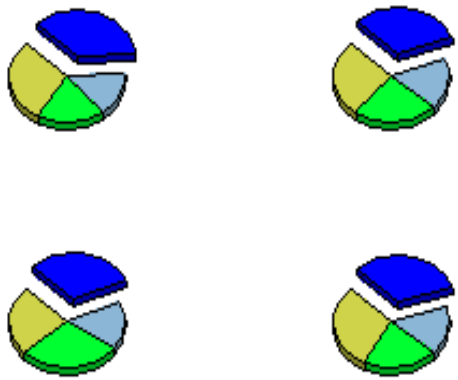


Figure 57 Exploding Pies

Starting Angle For Pies

By default, all pies are drawn starting at 135.0 degrees. By using `XtNxrtPieStartAngle`, users can specify the angle at which the pie should start drawing.

4.12 HiLo, Candle and Bubble Graphs

XRT/graph's HiLo, HiLoOpenClose, Candle and Bubble graph types use the Y-values in multiple sets to construct each bar or bubble. HiLo and bubble graphs use every two sets and HiLoOpenClose and candle graphs use every four sets.

Each group of sets makes up a "logical set" that you use to specify set labels, markers, text areas, and so on. Almost every XRT/graph resource, procedure, or method uses *logical* sets when the graph type is HiLo, HiLoOpenClose, candle or bubble. The only exceptions are the data object itself (in order to access the values in the sets comprising the bar or bubble), and data styles (to customize a portion of the bar or bubble in complex candle and bubble charts).

Bubble Resources

For bubble graphs, the `XtNxrtBubbleMethod` resource determines the display of the bubble value. The following are the possible values that the `XtNxrtBubbleMethod` resource can have:

`XRT_BUBBLE_METHOD_DIAMETER`	Displays information by varying the diameter of the bubble value. This is the default.

XRT_BUBBLE_METHOD_AREA	Displays information by varying the area of the bubble value.

Three values are required for each point in a bubble graph. The X- and Y-values for each point in the logical set are taken from the X- and Y-values of the corresponding points in the first physical set. The third value for each point is taken from the Y-value of the corresponding point in the second physical set. The X-value in the second physical set is ignored. The point assembled in this fashion is referred to as the *Bubble value.*

The resources XtNxrtBubbleMin and XtNxrtBubbleMax control the minimum and maximum diameter/area of the point symbols. These resources are XrtFloat values specified as a percentage of the diameter/area of the plot area. The diameter of the plot area is defined as the smaller of the plot area height or the plot area width. The area of the plot area is defined as height multiplied by width. The two resources are used in conjunction with the smallest and largest bubble values in the chart group to determine the diameter/area of the symbol drawn at each point. When the value of XtNxrtBubbleMin is larger than the value of XtNxrtBubbleMax, the point symbol diameter/area decreases as bubble values increase. Bubble values do not need to be strictly positive or strictly negative, since the diameter/area of point symbols is only dependent on the range of bubble values.

Lines are drawn between Bubble values as they are in plot graphs. The line pattern, line color and line width for each logical set are taken from the data style associated with the first physical set, although most bubble charts are best expressed without lines between the points. The point style determines the basic shape of the bubble value. This will default to a triangle, although most bubble charts are best expressed using the circular point. These issues are solved by setting the line pattern to XRT_LPAT_NONE and the point style to XRT_POINT_DOT. The point color, background color and fill pattern from the data style combined with XtNxrtGraphShowOutlines, XtNxrtGraphForegroundColor and the bubble value determine the remaining characteristics of the point symbol. Which of these items that are actually used, depends to some extent on the point style:

How the Symbol Is Drawn

XRT_POINT_DOT XRT_POINT_BOX XRT_POINT_TRI XRT_POINT_DIAMOND XRT_POINT_INVERT_TRI	■ Symbol drawn will be filled using the fill pattern, point color and background color.

XRT_POINT_STAR XRT_POINT_CROSS XRT_POINT_SQUARE XRT_POINT_CIRCLE XRT_POINT_VERT_LINE XRT_POINT_HORIZ_LINE XRT_POINT_DIAG_CROSS XRT_POINT_OPEN_TRI XRT_POINT_OPEN_DIAMOND XRT_POINT_OPEN_INVERT_TRI	■ Lines composing the symbols drawn using the point color. ■ Width of the lines equal to diameter of the symbol divided by 7.

How Diameter and Area are Interpreted

XRT_POINT_DOT XRT_POINT_BOX XRT_POINT_TRI XRT_POINT_DIAMOND XRT_POINT_INVERT_TRI XRT_POINT_SQUARE XRT_POINT_CIRCLE XRT_POINT_OPEN_TRI XRT_POINT_OPEN_DIAMOND XRT_POINT_OPEN_INVERT_TRI	■ Diameter defined as the height of the symbol. ■ Area defined as the actual area of the figure.
XRT_POINT_STAR XRT_POINT_CROSS XRT_POINT_VERT_LINE XRT_POINT_HORIZ_LINE XRT_POINT_DIAG_CROSS	■ Diameter defined as the length of one of the lines. ■ Area defined as the area of a circle with the above diameter.

HiLoOpenClose Resources

When the graph type is `XRT_TYPE_HILO_OPEN_CLOSE`, several resources control how open and close ticks are displayed:

`XtNxrtHiLoOpenShow`	Displays or hides open tick marks
`XtNxrtHiLoCloseShow`	Displays or hides close tick marks
`XtNxrtHiLoOpenCloseFullWidth`	Displays open/close ticks across the full width of the bar. This is useful for creating error bar charts.

XtNxrtCandleComplex

XRT/graph supports two styles of candle graphs, specified using the `XtNxrtCandleComplex` resource. When set to `False`, the datastyle from the first set determines the appearance of the candle. A transparent candle indicates a rising stock price, and a filled candle indicates a falling stock price. The color, width, line pattern, fill pattern, and point size elements are used.

When set to `True`, the datastyles in all four sets are used for the appearance:

- First datastyle color, width, and line pattern elements are used for the HiLo line.
- Second datastyle color, width, and fill pattern elements are used for the candle of a rising price.

- Third datastyle color, width, and fill pattern elements are used for the candle of a falling price.
- Fourth datastyle color is used for the candle outline.

See section 4.9 on page 74 for details on customizing data styles.

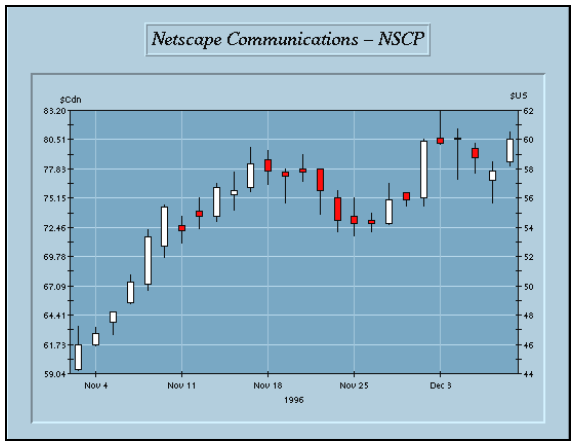


Figure 58 Complex Candle graph

4.13 Special Polar/Radar Graph Resources

Polar, radar, and filled radar graphs have several resources that control the handling and placement of the X-origin and Y-annotation. In these graph types, the X-axis is a circle, and the Y-axis measures distance from the center of the circle.

Angle Unit

The `XtNxrtAngleUnit` resource controls the units that the graph uses to interpret X-values and X-axis resource values. Degrees are used by default. Set this resource to `XRT_ANGLE_RADIANS` if your X-values should be interpreted in radians, or `XRT_ANGLE_GRADS` to use grads.

X-axis Base Origin

The "origin" of the X-axis is the circle's zero-angle, located on polar graphs by default at *3 o'clock* on the circle, and on radar/filled radar charts at *12 o'clock*. Use the `XtNxrtXOriginBase` resource to specify the location of the zero-angle. The value specified is interpreted in the units specified by `XtNxrtAngleUnit`.

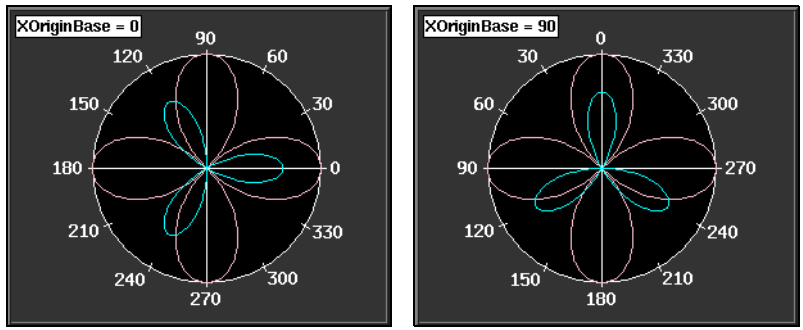


Figure 59 X-axis origin base at zero and 90 degrees

Annotation Angle

Y-axis annotation is drawn by default along the Y-axis at the X-origin value. You can position the annotation at any point on the circle using the `XtNxrt[YY2]AnnotationAngle` resources.

Polar X-axis Range

The `XtNxrtPolarHalfRange` Boolean resource enables you to change from a full-range polar graph (0-360 degrees) to a half-range graph (-180 to 180 degrees).

Polar Negative Values

The `XtNxrtPolarAxisAllowNegatives` Boolean resource permits or denies the ability to have negative values (origins) for polar graphs.

4.14 Foreground and Background Colors

Background Colors

The window background color is specified through the `XtNxrtBackgroundColor` resource. It is white by default.

Each of the header, footer, legend and graph areas also have a background color which is `NULL` (transparent) by default. For example, the resource to change the legend background color is `XtNxrtLegendBackgroundColor`. The data area of the graph also has its own background color, specified by `XtNxrtDataAreaBackgroundColor`.

When running on a monochrome Server, all non-black background colors will display as white.

Foreground Colors

The window foreground color is black by default, and is specified with the `XtNxrtForegroundColor` resource. It will be used as the foreground color for each of the header, footer, legend and graph areas, unless a different foreground color is specified for one or more of these areas. For example, to specify a different header

foreground color, use the XtNxrtHeaderForegroundColor resource. The data area of the graph also has its own foreground color, specified by XtNxrtDataAreaForegroundColor.[1]

When running on a monochrome Server, all non-white foreground colors will display as black.

Specifying F/B Colors

All foreground and background resources take a color string (as described in section 2.8 on page 29) as their value. Pastel colors such as "LightBlue", and "Wheat" often make good background colors. Saturated colors are often good foreground colors. For example:

```
XtVaSetValues(my_graph,
    XtNxrtBackgroundColor, "LightBlue",
    XtNxrtForegroundColor, "NavyBlue",
    XtNxrtHeaderBackgroundColor, "LightGray",
    XtNxrtFooterForegroundColor, "DarkGreen",
    NULL);
```

Take care not to choose a background color that is also used to display data in the graph. The following colors will conflict with the default data style colors and should *not* be used if default data styles are used: aquamarine, blue, cadet blue, gold, green, green yellow, khaki, light blue, light sky blue, lime green, magenta, medium blue, medium orchid, medium turquoise, medium violet red, plum, olive drab, orange, red, sky blue, steel blue, tan and yellow green.

4.15 Markers

Markers provide an application with a way to identify a particular data element on the screen, or a particular value in a plot. There are many potential uses for markers, but the most likely one is to provide feedback to the application user when a particular part of a graph is selected with the mouse.

Plot, area, bar, stacking bar, HiLo, HiLoOpenClose and candle graphs support the notion of both an X- and Y-marker. On each of these types except stacking bar, the X-marker is a line drawn across the graph area perpendicular to the X-axis. On stacking bar graphs, the X-marker is a short line drawn through one of the bar blocks if the X-marker is XRT_MARKER_SET_POINT, otherwise it is a line across the graph area perpendicular to the X-axis. The Y-marker is a line drawn across the graph area perpendicular to the Y-axis.

Pie, polar, radar, and filled radar charts support the notion of just one marker: a line drawn from the center of the chart and extending outward through the center of the selected data point or pie slice.

For combination graphs (see section 7.2 on page 128), specify the marker dataset using XtNxrtMarkerDataset.

1. This resource is ignored when XtNxrtGraphShowOutlines is False.

Marker Data Styles

By default, the markers are a black, 1-pixel wide, dashed line. Use the `XtNxrtMarkerDataStyle` resource to set the style for *both* markers, or use the `XtNxrt[XY]MarkerDataStyle` resources to set the style for *each individual* marker. Markers use the line pattern, color, and line width elements of the `XrtDataStyle` structure (see section 4.9 on page 74 for details on setting data styles).

`XtNxrtMarkerDataStyle` is used unless you explicitly set one of the `XtNxrt[XY]MarkerDataStyle` resources. Then `XtNxrt[XY]MarkerDataStyle` is used for that marker. Set the corresponding `UseDefault` resource to `True` to restore default behavior.

Marker Performance

Updating the marker positioning resources occurs very quickly. This is because XRT/graph does not repaint the entire graph window to reposition the markers. Marker performance is improved by setting `XtNxrtDoubleBuffer` to `True`. Generally though, graph performance is improved by turning double buffering off.

Depending on the performance of your CPU, network and X Server, it may be possible to track mouse-dragging in real time with markers. The *feedback* and *scroll_time* demo programs provide examples of mouse tracking using markers.

X-marker Positioning

The `XtNxrtXMarkerMethod` resource controls the positioning method used for the X-marker. It can be set to the following values:

`XRT_MARKER_SET_POINT`	The `XtNxrtXMarkerSet` and `XtNxrtXMarkerPoint` resources specify the X-marker position. This is the only method you can use for pie, radar, and filled radar charts.
`XRT_MARKER_VALUE`	The `XtNxrtXMarker` resource positions the X-marker.

`XtNxrtXMarkerSet` and `XtNxrtXMarkerPoint` are indices into the data object. Together, they define the particular data point through which the X-marker will be drawn.

The `XtNxrtXMarker` floating-point resource is the X-axis value through which the marker will be drawn.

Y-marker Positioning

The Y-marker is positioned through the `XtNxrtYMarker` floating-point resource. It specifies the value on the Y-axis through which the Y-marker will be drawn. This resource is ignored on pie, polar, radar, and filled radar charts.

Neither X- nor Y-markers will appear unless the appropriate `XtNxrt[XY]MarkerShow` resource is set to `True`.

4.16 Area Borders

Each of the 4 graph areas (header, footer, legend and graph) may be enhanced with a border. There is a Border and a Border Width resource for each of these areas (for example, `XtNxrtGraphBorder` and `XtNxrtGraphBorderWidth`). In addition, you can specify the border for the entire widget, using `XmNshadowThickness` to control its width.

The Border resources may be set to any of: `XRT_BORDER_NONE`, `XRT_BORDER_PLAIN`, `XRT_BORDER_3D_OUT`, `XRT_BORDER_3D_IN`, `XRT_BORDER_ETCHED_IN`, `XRT_BORDER_ETCHED_OUT`, `XRT_BORDER_FRAME_IN`, `XRT_BORDER_FRAME_OUT`, `XRT_BORDER_BEVEL` or `XRT_BORDER_SHADOW`. The width must be between 0 and 20.

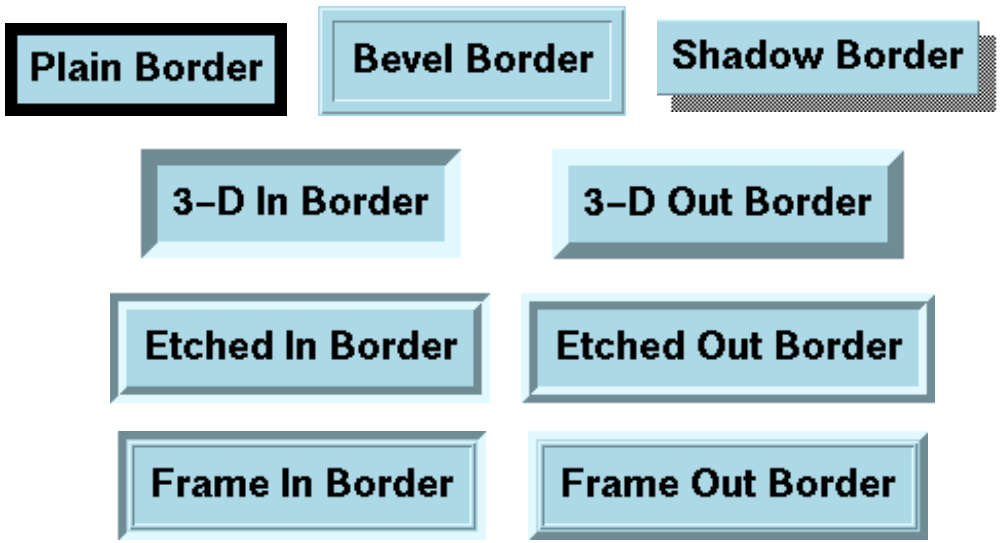


Figure 60 Border Types

4.17 Background Images

The header, footer, legend, and graph, and data areas may be enhanced with a background image. There is an Image and an Image Layout resource for each of these areas (for example, `XtNxrtGraphImage` and `XtNxrtGraphImageLayout`). In addition, you can specify an image for the entire widget.

The Image resources are of type XrtImage. The XrtImage type is actually the same as the XrtPDSImage type discussed in section 8.1 on page 143.

The following example shows how you can use XRT/graph's converters to set a background image in the footer:

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

void
set_image(Widget graph, 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(graph, XtRString, &from, XtRXrtImage, &to)) {
    fprintf(stderr, "Conversion of %s to XtRXrtImage failed\n",
            file_name);
    image = NULL;
}
XtVaSetValues(graph,
              XtNxrtFooterImage,    image,
              NULL);
    XrtPDSImageDestroy((XrtPDSImage)image);
}
```

Note that images are copied by the widget and thus you must destroy the image that was created by the converter. To remove an image, set the corresponding image resource to `NULL`.

An alternative way to set an image is to set the image property via a string using *XrtSetPropString()*.

```
    XrtSetPropString(graph, XRT_FOOTER_IMAGE,
                     "$XRTHOME/src/icons/shading.xpm");
```

This function uses XRT/graph's converters directly, bypassing *XtConvertAndStore()*. All the work of setting the image on the widget is done internally. The *$XRTHOME* environment variable is expanded (as it would be if the image was specified in a resource file).

For more information on which image types are supported and how you can get XRT/graph to support custom image types see section 8.1 on page 143.

The Image Layout resources specify how to size and position the background image. The following layouts are available:

- `XRT_IMAGELAYOUT_CENTERED` places the image as-is in the center of the area.
- `XRT_IMAGELAYOUT_TILED` displays the image tiled to cover the entire area.

When an image is specified for an area, it supersedes the background color for the area. However, if a background color has been specified for the header, it will overwrite an image specified for the widget.

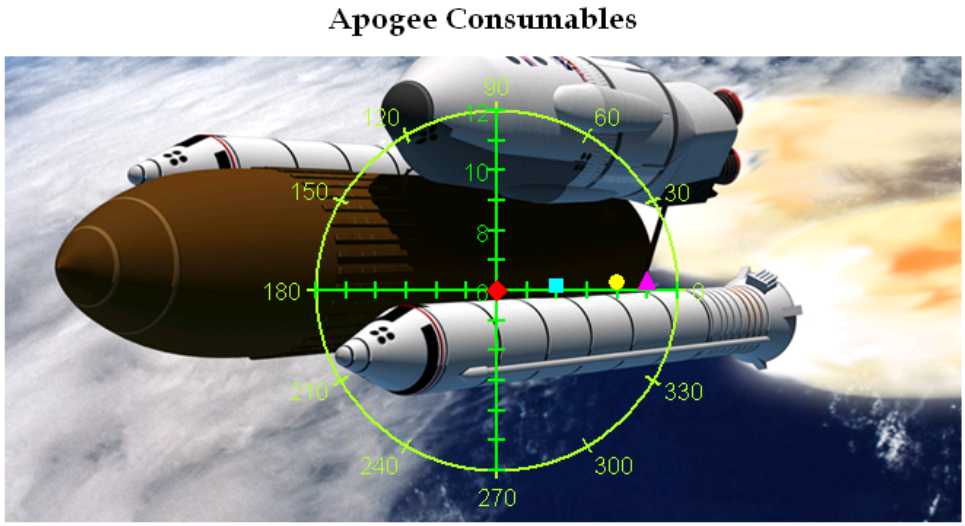


Figure 61 Graph background image

Minimum Image Size

This feature allows applications to use background images in text areas, headers and footers as labels without requiring text. This is achieved by setting one of `XtNxrtTextImageMinimumSize`, `XtNxrtHeaderImageMinimumSize` or `XtNxrtFooterImageMinimumSize`. When these (the appropriate one for the appropriate area) are set to `True`, they give the label a minimum size equal to the size of the image.

One possible use for this would be to place a customized point anywhere within a graph by using the text area feature in combination with `XtNxrtTextImageMinimumSize`.

4.18 Double Buffering

Double buffering is a graphics technique which will reduce the amount of flashing perceived by a user when a graph changes.

When `XtNxrtDoubleBuffer` is `True`, every time XRT/graph changes a graph it will:

- allocate (if necessary) and clear a Pixmap in the Server.
- render the complete graph to the Pixmap.
- tell the Server to copy the Pixmap to the screen.

When `XtNxrtDoubleBuffer` is `False`, every time XRT/graph changes a graph it will clear the screen image (possibly causing a visual flash) and then render the complete

image to the visible window (possibly allowing the user to see the graph being drawn piece by piece).

By default, `XtNxrtDoubleBuffer` is `True`. However, setting it to `False` can improve the graphing performance and reduce memory requirements on the Server.

4.19 Adding Scrollbars to a Graph

A graph widget can be located in a ScrolledWindow widget. The ScrolledWindow widget provides the user with scrollbars so that graphs which are too big to fit in the window can still be viewed. The ScrolledWindow widget takes care of all scrolling details and repositions the graph automatically.

The following code segment illustrates how to do this. First, a `ScrolledWindowWidget` is created with `XmNscrollingPolicy` set to `XmAUTOMATIC`. Then the graph is created. Finally, the ScrolledWindow's `XmNworkWindow` resource is set to the graph.

```
Widget      scrollw, graph;
scrollw = XtVaCreateManagedWidget("scrollw",
    xmScrolledWindowWidgetClass, toplevel,
    XmNscrollingPolicy, XmAUTOMATIC,
    NULL);
graph = XtVaCreateManagedWidget("my_graph",
    xtXrtGraphWidgetClass, scrollw,
    XtNwidth, 900, /* Set graph size here */
    XtNheight, 400,
    NULL);
XtVaSetValues(scrollw,
    XmNworkWindow, graph,
    NULL);
```

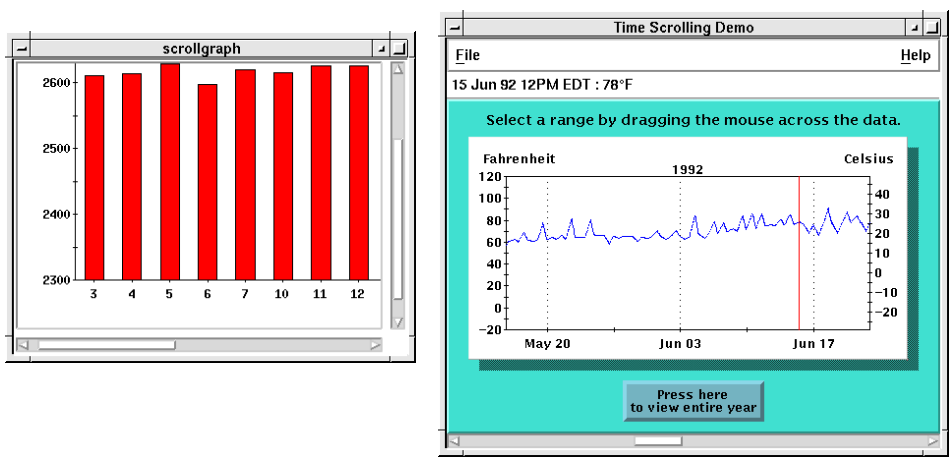


Figure 62 Two ways to use scrollbars with a graph

A scrolling graph example (*scrollgraph.c*) can be found in *$XRTHOME/src/graph/examples*.

Often it is better to use scrollbars to control the scrolling of data displayed in a graph rather than the entire graph. An example of this can be found in the *scroll_time* demo program, located in *$XRTHOME/src/graph/demos/scroll_time*.

4.20 Output and Printing

Many applications need to provide the user with a way to get a hardcopy of a graph. XRT/graph provides several procedures which may be used to output graph representations to files or printers.

XRT/graph provides several methods to assist developers in the printing of XRT/graph windows. These methods will work correctly even when the graph window is obscured by other windows. The methods are fully documented in Appendix D on page 211.

XrtDrawPS()

This procedure is recommended for creating PostScript descriptions of the graph. This routine creates an Encapsulated PostScript file using native PostScript operators. There are parameters for font selection, color handling and graph sizing and positioning. The procedure is fully documented in Appendix D on page 211.

XrtOutputPS()

Like *XrtDrawPS()*, *XrtOutputPS()* also creates an encapsulated PostScript description of the graph, however, it does not use native PostScript operators. Instead, using the image or colorimage operators, it creates a PostScript bitmap equivalent to the image on the screen. This can result in a very large print file which may not take advantage of the full printer resolution. For a complete description of this procedure, see Appendix D on page 211.

XrtOutputRaster()

XrtOutputRaster() creates a Sun rasterfile format description of the graph. This procedure is only available on SPARC architectures. For a complete description of this procedure, see Appendix D on page 211.

XrtOutputXwd()

XrtOutputXwd() creates an X Window Dump format description of the graph. For a complete description of this procedure, see Appendix D on page 211.

XrtGetImage()

XrtGetImage() returns a pointer to an *XImage* representation of the graph. This may be used by developers who wish to implement their own unique output formats. It is documented in Appendix D on page 211.

Note: You can use the image routines, such as *XrtSaveImageGeneric(),* to output a graph as other types of images, such as JPEG, PNG and user-defined types. See "XrtPDSImage" on page 143.

Sample Print Pop-up

The source code for a sample print pop-up window which shows one way to use *XrtDrawPS()* and how to pipe the file to *lp* is in *$XRTHOME/src/tools/pdbox_graph.c.*

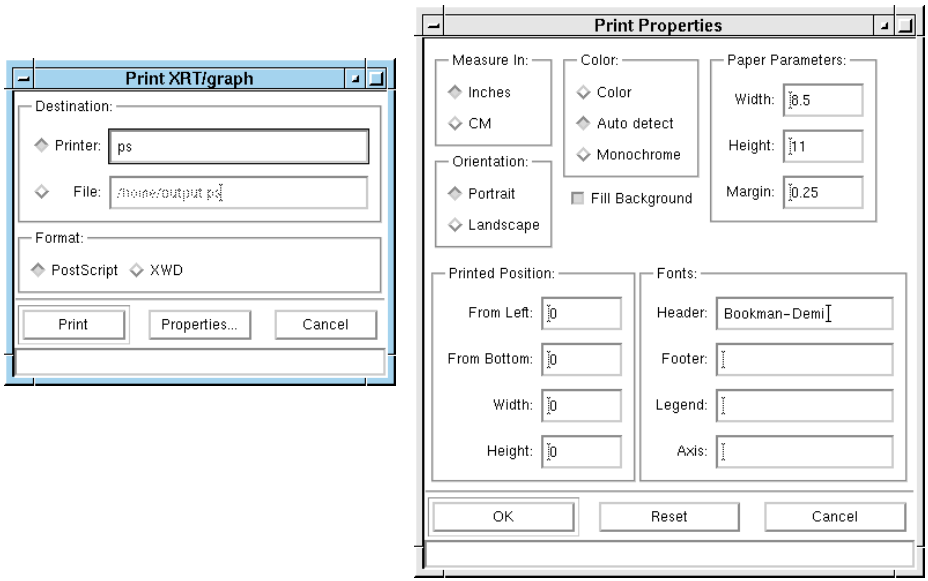


Figure 63 Print Pop-ups created by the Sample Print Pop-up Code

5

XRT/graph Data

Getting Data into Graphs ■ *The Data Object*
Object and Data Access Methods ■ *Excluding and Hiding Data*
Transposing the Data ■ *Using Multiple Datasets*
Example Using Static Data ■ *Example Reading Real-time Data from a Pipe*

5.1 Getting Data into Graphs

Data to be displayed can originate from diverse sources: Unix files, databases, real-time data feeds, or even unrelated processes running on other machines.

Perhaps the most difficult task facing an XRT/graph developer is retrieving the data to be displayed, and inserting it into a data object (referenced by `XrtDataHandle`) for displaying. This chapter discusses the data object in detail and offers several examples of allocating and loading it.

Data from a File

If the data exists in a Unix file (and it does not need to be changed or updated in real-time) there are two options to consider. The first option is to massage the data so that the file conforms to a syntax understood by *XrtDataCreateFromFile()*. This procedure will allocate the data object, load it with data from a named file, and return a handle to the data. *XrtDataCreateFromFile()* and the data file syntax is documented in Appendix D on page 215.

Another approach is to allocate a data object using *XrtDataCreate()*, and populate it with data by reading the data file (perhaps using *fgets(3)* or *fscanf(3)*). When the data object is loaded, the graph can be created and the `XtNxrtData` resource set to the `XrtDataHandle` that references it.

Real-Time Data

Data that changes in real-time, and graphs that need to be updated in real-time, require more careful coding. The Xt Intrinsics provides functions which can be used to trigger application callback routines to deal with real-time events.

In particular, *XtAppAddInput()* can be used to register an application callback routine to be called whenever there is input pending on a file descriptor. The file descriptor can represent a file, pipe or even a socket connection to a process running on another machine.

Another approach is for the application to use *XtAddTimeOut()* to define an interval timer. Whenever the specified time interval expires, the application's callback routine will be executed. At this time, the application can check to see if there is any more data to graph and take action accordingly.

Your Xt Intrinsics documentation should be consulted for more information about these routines.

Fast Update Procedures

For some types of real-time applications, particularly when creating a strip-chart that has data continually being added to it, or a scatter plot which is receiving new points, the Fast Update procedures will be helpful. See section 7.6 on page 135 for more information.

5.2 The Data Object

When XRT/graph begins to draw a graph, it uses the data values located in the data object pointed to by the `XtNxrtData` resource.[1]

An `XrtDataHandle` is a handle to a data object. You change the data contained in the data object and the object's attributes by using a collection of methods. These methods reference the data object through its `XrtDataHandle`. Using data objects insulates your application from future changes to the internal data structure and imposes no speed or memory penalty on your application.

Primary Dataset

A second data object can be displayed in the same graph using the `XtNxrtData2` resource. However, when `XtNxrtData` is `NULL` and `XtNxrtData2` points to a valid `XrtDataHandle`, XRT/graph uses the data referenced by `XtNxrtData2` as its *primary* dataset. Y2-axis resources are used (Y2 is the primary Y-axis), along with `XtNxrtType2`, `XtNxrtSetLabels2`, `XtNxrtPointLabels2`, and `XtNxrtDataStyles2`.

Responsibility for Data

The application is responsible for allocating and destroying all required data objects. XRT/graph does not make a copy of the data, instead it references data in the application's memory which is pointed to by `XtNxrtData`.

If XRT/graph has to repaint the graph window, it will use the data pointed to by `XtNxrtData`.

1. This resource can still be set to point to an `XrtData` structure, for compatibility with previous releases of XRT/graph. However, several data object attributes cannot be set using the API used with `XrtData` structures.

5.2.1 Array and General Data Formats

The data stored in a data object can be in either *Array* or *General* format.

Array Data

Array data is defined by sets of Y-values sharing common X-values. All sets contain the same number of points. X- and Y-values are of type `float`. Use Array data for bar, stacking bar, pie, HiLo, HiLoOpenClose, candle, radar, and filled radar charts; also for other graph types when they contain common X-values.

Array — 2 Sets of 3 Points:

	Point 0	Point 1	Point 2
Common X-values	1	2	3
Set 0 Y-values	150	175	160
Set 1 Y-values	125	100	200

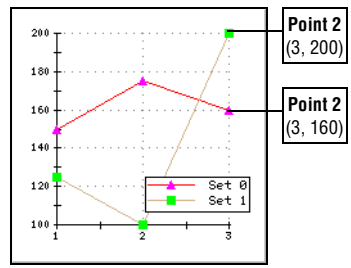


Figure 64 Example data and graph using Array data

General Data

General data is defined by sets of X-values and Y-values. Each set can contain a different number of points. X- and Y-values are of type `float`. General data can only be used for plot, area, polar and bubble graphs.

Note: Each set within a logical set should have the same number of points.

General — 2 Sets, Maximum of 4 Points:

		Point 0	Point 1	Point 2	Point 3
Set 0 4 Points	**X-values**	1	1.25	1.75	3
	Y-values	120	140	160	170
Set 1 2 Points	**X-values**	1.5	2.5		
	Y-values	130	190		

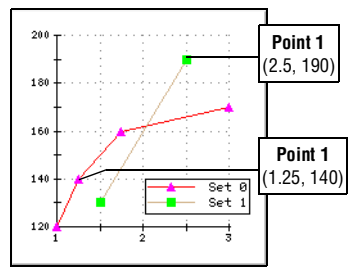


Figure 65 Example data and graph using General data

5.2.2 Basic XrtDataHandle Attributes

In addition to containing the graph data, the data object has the following basic attributes that can be modified:

Type — Specifies whether data is in Array or General format. *XrtDataGetType()* retrieves the current format. To change Array to General, call *XrtDataSetType()*. You cannot change General data to Array.

NSets — Specifies the total number of sets in the dataset. Use *XrtDataGetNSets()* to retrieve the current value, and *XrtDataSetNSets()* to change it.

NPoints — Specifies the total number of points in all sets (Array) or in one set (General). Use *XrtDataGetNPoints()* to retrieve the current value, and *XrtDataSetNPoints()* to change it.

Hole — Specifies the Y-value to interpret as missing data, that is, a hole (illustrated in Figure 66). The default hole value is `XRT_HUGE_VAL`. Use *XrtDataGetHole()* to retrieve the current hole value, and *XrtDataSetHole()* to change it.

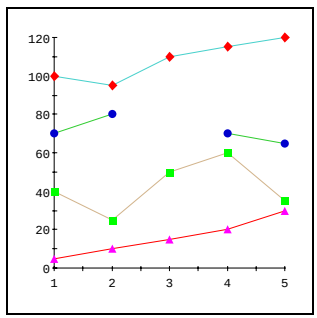


Figure 66 Example of a Data Hole in a Plot

5.3 Object and Data Access Methods

There are a number of procedures and methods that set and retrieve data inside a data object, and other methods that manipulate data objects themselves. They are summarized in the following table. Complete reference information is in Appendix D on page 211. The source code for many of the convenience procedures is in *$XRTHOME/src/graph/util*.

Adding and Removing Sets/Points	
XrtArrDataAppendPts()	Appends a column of points to an Array data object.
XrtGenDataAppendPt()	Appends an (x,y) value to the end of a set of data in a General data object.

XrtArrDataRemovePts()	Removes a point from each set in an Array data object.
XrtGenDataRemovePt()	Removes a point from a General data object.
XrtDataRemoveSet()	Removes a set from a data object.
Retrieving and Changing Data Values	
XrtDataGetXData() *XrtDataGetYData()*	Retrieves a pointer to a set's *internal* list of X- or Y-values. Disables other data changes until *XrtDataRelease()* is called.
XrtDataGetXElement() *XrtDataGetYElement()*	Returns the X- or Y-value at a set and point index.
XrtDataRelease()	Returns control of the data object to XRT/graph.
XrtDataSetXData() *XrtDataSetYData()*	Sets a list of X- or Y-values in a set.
XrtDataSetXElement() *XrtDataSetYElement()*	Sets the X- or Y-value at a set and point index.
XrtDataSort()	Sorts the points in each set by increasing X-value.
XrtDataUpdateDataValue()	Changes a data value to a new value throughout a data object.
Manipulating Data Objects	
XrtDataConcat()	Creates a new data object by concatenating the data pointed to by two existing data objects.
XrtDataConvertToHandle()	Converts an `XrtData` structure from previous releases to a data object.
XrtDataCopy()	Creates a copy of a data object.
XrtDataCreate()	Creates a new data object.
XrtDataCreateFromFile()	Creates a new data object with data from a file.
XrtDataDestroy()	Destroys a data object.
XrtDataExtractSet()	Creates a new data object loaded with one set of data from an existing data object.
XrtDataSaveToFile()	Writes data in a data object to a file.
XrtDataTranspose()	Transposes set and point data in a data object.

The following code uses some of the data methods to output the data values in Array or General data objects.

```
static void
print_arr(XrtDataHandle d)
{
/* Print out the data contained in XRT_DATA_ARRAY type data object */
int i,j;

    printf("\nPrinting Array data with %d sets, %d points\n",
        XrtDataGetNSets(d), XrtDataGetNPoints(d, NULL));
    printf("X-Values:\n");
    for (j = 0; j < XrtDataGetNPoints(d, NULL); j++)
        printf("%f\t", XrtDataGetXElement(d, NULL, j));
    for(i = 0; i < XrtDataGetNSets(d); i++) {
        printf("Y %d Values:\n", i);
        for (j = 0; j < XrtDataGetNPoints(d, NULL); j++)
            printf("%f\t", XrtDataGetYElement(d, i, j) );
        printf("\n");
    }
}

static void
print_gen(XrtDataHandle d)
{
/* Print out the data contained in XRT_DATA_GENERAL type data object */
int i,j;

    printf("\nPrinting General data with %d sets\n", XrtDataGetNSets(d));
    for(i = 0; i < XrtDataGetNSets(d); i++) {
        printf("Set %d Values:\n", i);
        for (j = 0; j < XrtDataGetNPoints(d, i); j++)
            printf("(%f, %f)\t",
                XrtDataGetXElement(d, i, j), XrtDataGetYElement(d, i, j) );
        printf("\n");
    }
}

static void
pr_data(XrtDataHandle d)
{
    if ( XrtDataGetType(d) == XRT_DATA_ARRAY )  /* is array */
        print_arr(d);
    else                    /* is general */
        print_gen(d);
}
```

Figure 67 Data object output routines

Direct Data Access

Normally an application does not access the memory used by a data object directly. Most of the data object methods copy values you pass in to the object's internal data structure.

However, XRT/graph provides special methods that you can use to change several X- or Y-values as quickly as possible. *XrtDataGetXData()* and *XrtDataGetYData()* return the *internal* pointer to a set's array of X- or Y-values. Your application can change the floating-point values in this array directly. No other data changes to the

object are allowed until you return ownership of the array back to XRT/graph by calling *XrtDataRelease()*.

The following example uses these methods to increase or decrease the X- and Y-values in a set by a given amount value:

```
static void
delta_set(XrtDataHandle d, int set, float xdel, float ydel)
{
    /* Add xdel and ydel to all points in set */
    int     i, npoints;
    float   *xdata, *ydata;
    if ((npoints = XrtDataGetNPoints(d, set)) < 0) {
      return;
    }

    xdata = XrtDataGetXData(d, set);
    ydata = XrtDataGetYData(d, set);
    if (xdata && ydata) {
        for (i = 0; i < npoints; i++) {
            xdata[i] += xdel;
            ydata[i] += ydel;
        }
    }
    XrtDataRelease(d);

    XtVaSetValues(graph, XtNxrtData, d, NULL);
}
```

5.4 Excluding and Hiding Data

By default, all of the data contained in a data object is graphed. The object's attributes allow you to graph just a range of sets or points, and even to temporarily hide or exclude any particular set from the graph.

Defining Beginning and Ending Set and Point

The data object can graph a subset or range of sets and points from the data contained within it. This can be useful, for example, to display current data only while retaining but not displaying historical data in the data object. You can specify the first and last set to display and also the first and last point using the following data object attributes:

FirstSet — Specifies the index of the first set to graph. Use *XrtDataGetFirstSet()* to retrieve the current value, and *XrtDataSetFirstSet()* to change it.

LastSet — Specifies the index of the last set to graph. Use *XrtDataGetLastSet()* to retrieve the current value, and *XrtDataSetLastSet()* to change it.

FirstPoint — Specifies the index of the first point to graph in all sets (Array) or in one set (General). Use *XrtDataGetFirstPoint()* to retrieve the current value, and *XrtDataSetFirstPoint()* to change it.

LastPoint Specifies the index of the last point to graph in all sets (Array) or in one set (General). Use *XrtDataGetLastPoint()* to retrieve the current value, and *XrtDataSetLastPoint()* to change it.

Hiding and Excluding Individual Sets

XRT/graph makes it easy to temporarily remove one or more sets from the graph display. The *XrtDataSetDisplay()* and *XrtDataGetDisplay()* procedures access the data object's Display attribute for any set. Valid values are:

`XRT_DISPLAY_SHOW` Set is displayed in the graph. This is the default value.

`XRT_DISPLAY_HIDE` Set is hidden from display but used in calculating the axes and position of other graph elements.

`XRT_DISPLAY_EXCLUDE` Set is excluded from the chart.

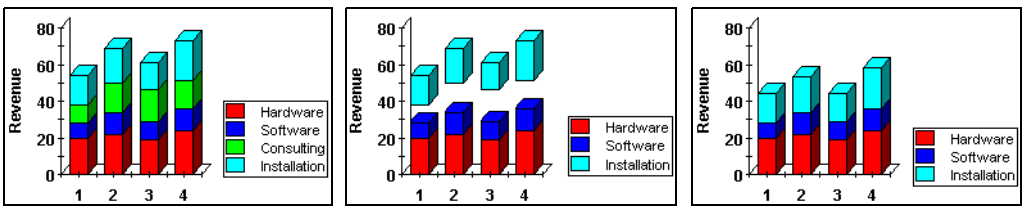


Figure 68 Set Display attribute set to _SHOW (left), _HIDE (middle), and _EXCLUDE (right)

Hide/Exclude Rules for HiLo/HiLoOpenClose/Candle graphs and Bubble Charts

For these graph types, excluding any set of a logical set excludes the entire logical set. Hiding either the high or low set hides the entire logical set. Hiding either the open or close set in a candle chart hides them both.

5.5 Transposing the Data

Transposing the graph means *switching* the sets and points. For example, the *mm63.dat* file listed in Figure 4 on page 13 has 2 sets, each with 4 points. When this data is transposed, it becomes 4 sets, each with 2 points. Not all datasets are meaningful when transposed.

Two methods provide different degrees of transposition. *XrtDataTranspose()* transposes the data values, and *MtGraphTransposeData()* transposes a graph completely, switching sets and points in the data, Set- and Point-labels, and marker and text area positioning.

Figure 69 shows the results of transposing a simple bar chart. With this particular dataset, a bar chart of the transposed data shows 2 groups (Revenue and Expenses) of 4 bars (one for each quarter). The original data shows 4 groups (one for each quarter) of 2 bars (Revenue and Expenses).

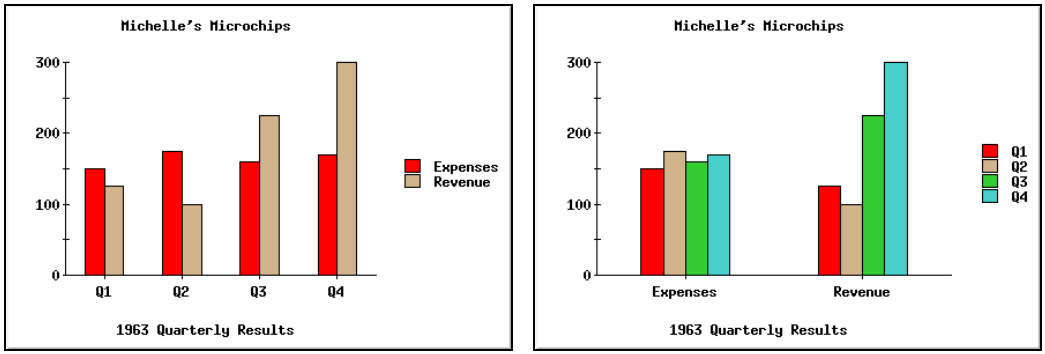


Figure 69 Normal (left) and Transposed (right) bar charts

MtGraphTransposeData() (source in *$XRTHOME/src/tools/mtgraph.c*)
Transposes the data, Set- and Point-labels, marker and text area attachments. To use this method, you must include *$XRTHOME/src/tools/mtgraph.h* and link your program to the XRT "tools" library.

```
Boolean
MtGraphTransposeData(
    Widget          graph,
    XrtDataHandle   data,
    XrtDataHandle   data2,
    Boolean         sw_labels,
    Boolean         sw_markers,
    Boolean         sw_text
    )
```

data is a new data object (already transposed); if `NULL`, the widget's current data is transposed. If not `NULL`, set the passed in object on the widget. *data2* is the second dataset in a combination graph. *sw_labels* specifies whether to switch Point-labels and Set-labels. *sw_markers* specifies whether to switch the values of `XtNxrtXMarkerPoint` and `XtNxrtXMarkerSet`. *sw_text* specifies whether to switch the set and point elements of any text areas attached by `XRT_TEXT_ATTACH_DATA`. This procedure returns `True` on success and `False` on failure.

For information on the *XrtDataTranspose()* procedure, see Appendix D on page 211.

5.6 Using Multiple Datasets

XRT/graph makes it easy to change entire datasets while keeping graph characteristics unchanged. Programs can create several data objects and then switch among them by setting the `XtNxrtData` or `XtNxrtData2` resources to point to the current data.

The code in Figure 70 sets the `XtNxrtData` resource to point to the appropriate data and updates the header text.

```
static char *dec_st[] = { "DEC", NULL, };

void
dec_proc(w, client_data, call_data)
Widget w;
XtPointer client_data;
XmToggleButtonCallbackStruct *call_data;
{
    if (call_data->set)
        XtVaSetValues(graph,
            XtNxrtData,         dec0990,
            XtNxrtHeaderStrings,dec_st,
            NULL);
}
```

Figure 70 Changing Datasets in a Callback Procedure

For an example of changing data, see the *stock.c* example program, located in *$XRTHOME/src/graph/examples*.

The strategy for changing data in real-time is just as simple. First, create a new data object and populate it with the new data (or update the data that is currently being displayed). Second, tell the graph to use the new data by setting the `XtNxrtData` resource.

5.7 Example Using Static Data

If the data to be displayed is known at compile time, it can simply be defined in static variables and then copied into a data object for displaying.

The following example defines the values to be displayed in static C variables `xdata[]` and `ydata[]`. The code simply allocates a data object of the correct size, and uses the data object methods to copy the static values into the structure.

```
#define NSETS 3
#define NPOINTS 5
static float xdata[NPOINTS] = {1, 2, 3, 4, 5,};     /* X-values */
static float ydata[NSETS][NPOINTS] = {
  {5, 8, 14, 19, 24},          /* Mexico */
  {11, 15, 18, 21, 22},        /* Tokyo */
  {14, 16, 15.5, 15.2, 15.4},  /* New York */
};
static char *sl[] = { "Mexico", "Tokyo", "New York", NULL, };
static char *pl[] = { "60", "70", "80", "90", "2000", NULL, };
...
    Widget graph, toplevel;
    XrtDataHandle pop;
    int i;

    /* Allocate and load the data */
    pop = XrtDataCreate(XRT_DATA_ARRAY, NSETS, NPOINTS);
    for (i=0; i<NPOINTS; i++) {
        XrtDataSetXElement(pop, NULL, i, xdata[i]);
    }
    for (i=0; i<NSETS; i++) {
        XrtDataSetYData(pop, i, ydata[i], NPOINTS, 0);
    }
...
    /* Create graph */
    graph = XtVaCreateManagedWidget("mygraph",
            xtXrtGraphWidgetClass,  toplevel,
            XtNxrtData,             pop,
            XtNxrtSetLabels,        sl,
            XtNxrtPointLabels,      pl,
            XtNxrtXAnnotationMethod, XRT_ANNO_POINT_LABELS,
            NULL);
```

Figure 71 Loading Data Object from Static C Variables

5.8 Example Reading Real-time Data from a Pipe

The *vmgraph* demonstration program is a good example of a program which reads real-time data from a pipe, parses the incoming data, and then graphs it. The complete source code for *vmgraph* is in the *$XRTHOME/src/graph/demos/vmgraph* directory.

After creating three graph objects, *vmgraph* forks a child process, which in turn starts up the *vmstat(8)* program. The standard output of the child process is set up to be the standard input (file descriptor 0) of the *vmgraph* process. *vmgraph* then uses *XtAddInput()* to tell the Xt Intrinsics to call the *read_line()* procedure whenever there is input pending on file descriptor 0:

```
XtAddInput(0, XtInputReadMask, read_line, NULL);
```

The code which forks the child process and sets up the pipe communication is reproduced below:

```
if (pipe(pipes) == -1) {
   fprintf(stderr, "pipe failed\n");
   exit(1);
}

switch(vmpid = fork()) {
   case -1:
       fprintf(stderr, "fork failed\n");
       exit(1);
   case 0: /* child, vmstat writes to pipe */
       close(pipes[0]);
       dup2(pipes[1], 1);
       sprintf(refr, "%d", sample_time);
       execl("/usr/ucb/vmstat", "vmstat", refr, 0);
       fprintf(stderr, "execl failed!\n");
       break;
   default: /* parent reads from pipe, waits for child */
       close(pipes[1]);
       dup2(pipes[0], 0);
       XtAppAddInput(app_context, 0, XtInputReadMask, read_line, NULL
       wait3(NULL, WNOHANG, NULL);
       break;
}
```

Figure 72 Forking a Process and Reading from it Through a Pipe

The *read_line()* routine reads a line of data from *vmstat*. After parsing it into individual data values, it calls routines which update each of the application's graph objects.

6

Programming User Interaction

Default User Interaction ■ *Overview of Translations and Actions*
Starting User Interaction ■ *Updating User Interaction*
Ending User Interaction ■ *Interacting with Graph Data*
Window Resizing

This chapter describes the user-interaction features of XRT/graph – how an end-user can interact with the graph and how an application can control that interaction.

6.1 Default User Interaction

By default, an end-user can scale, translate, and zoom into all types of graphs. Users can also rotate any bar, stacking bar, or pie chart displaying a 3D effect.[1] Figure 73 shows the graph interactions enabled by XRT/graph's default translations.

Property Editor

A right mouse button translation is also installed that calls the XRT/graph Property Editor. However, the Property Editor does not display unless the application specifically enables it. See section 3.1 on page 34 for details on enabling the Property Editor.

6.2 Overview of Translations and Actions

The graph's default translation table defines user events that cause some interactive action within the widget. You can customize the user's interaction capabilities through the following mechanisms:

1. 3D graphs must have a depth (`XtNxrtGraphDepth`) greater than zero to allow interactive rotation.

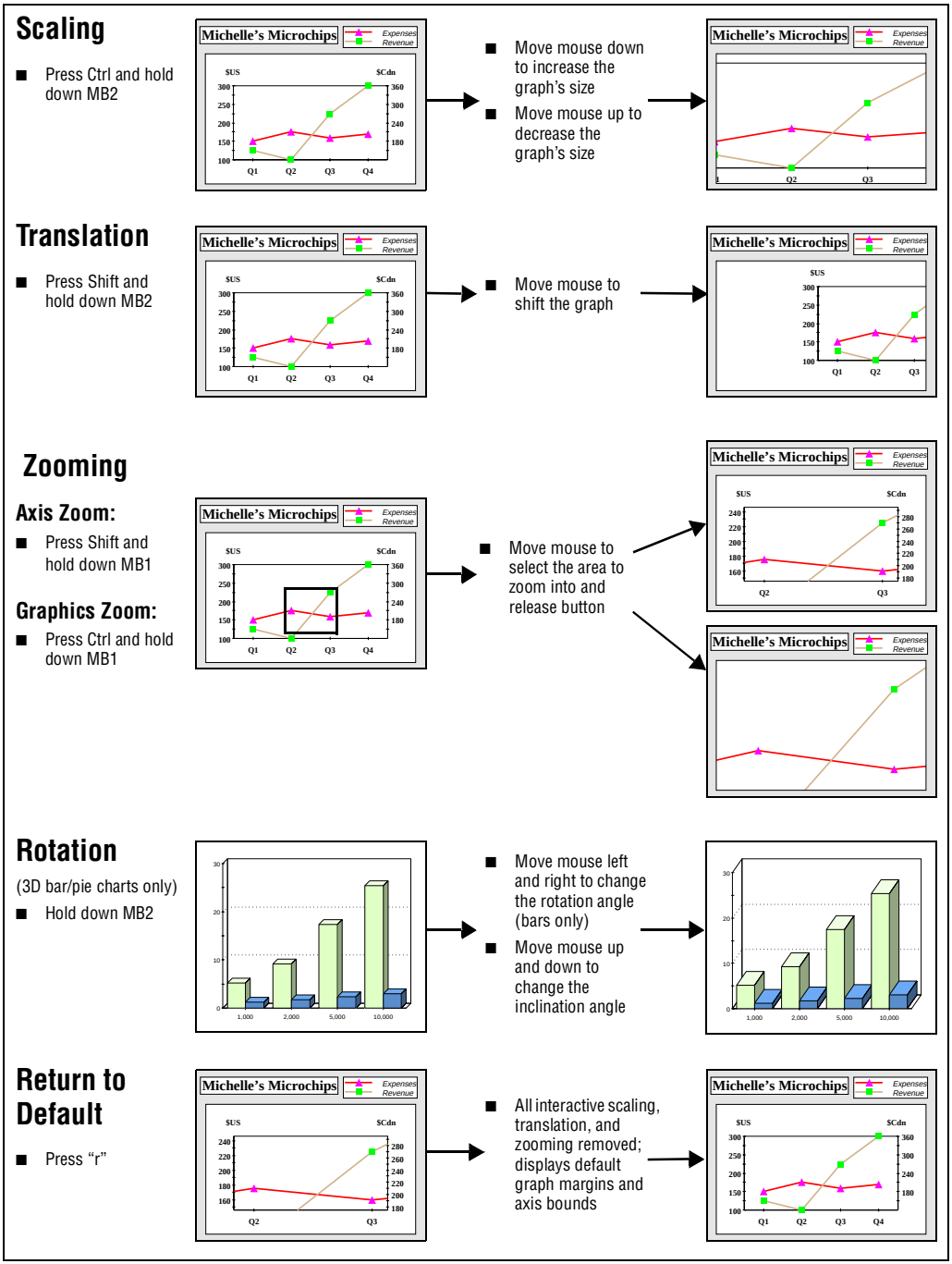


Figure 73 Default User-Interaction Translations

- Translation tables – An application can override XRT/graph'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. Appendix E on page 251 lists XRT/graph's default translations and actions.
- Callbacks – An application can be notified each time a user interacts with the widget by registering callback procedures that XRT/graph calls before, during, and after user interaction. 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 graph handles user interaction in three stages, each occuring 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.

6.3 Starting User Interaction

The *ModifyStart()* action is called to notify the application that a user interaction is about to begin. Figure 74 illustrates this action routine and shows where any Modify callbacks you add are called.

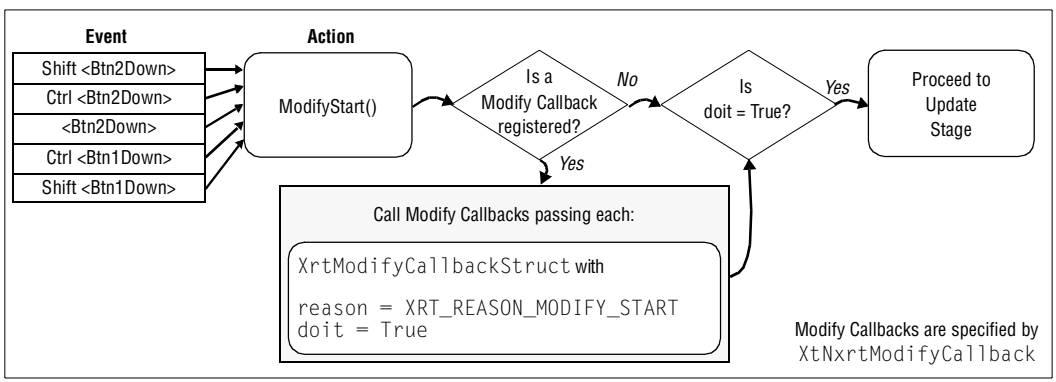


Figure 74 ModifyStart() Action Routine

Disabling All User Interaction

You can use `XtNxrtModifyCallback` to 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
    String   *params;   // Read-only
    Cardinal nparams;   // Read-only
    Boolean  doit;
} XrtModifyCallbackStruct;
```

Set the *doit* parameter of this structure to `False` (when the *reason* parameter is `XRT_REASON_MODIFY_START`) to disable all user interactions. When *doit* is `False`, *ModifyStart()* disables all "update" actions until the next *ModifyStart()* action is called.

6.4 Updating User Interaction

One of several action routines is called to notify the application that a user interaction is about to be updated. None of these routines updates the graph unless the interaction has successfully passed through the *ModifyStart()* action.

6.4.1 Scaling, Translation, and Graphics Zooming

The *ScaleUpdate()* action updates interactive scaling of the graph. The *TranslateUpdate()* action updates interactive translation of the graph. The *ZoomEnd()* action zooms into the graph at the area defined by the "zoom rubberband" (defined by the *ZoomUpdate()* action). These routines all alter the Graph Margin resources[1]. Figure 75 illustrates these action routines and shows where any Transform and Expose callbacks you add are called.

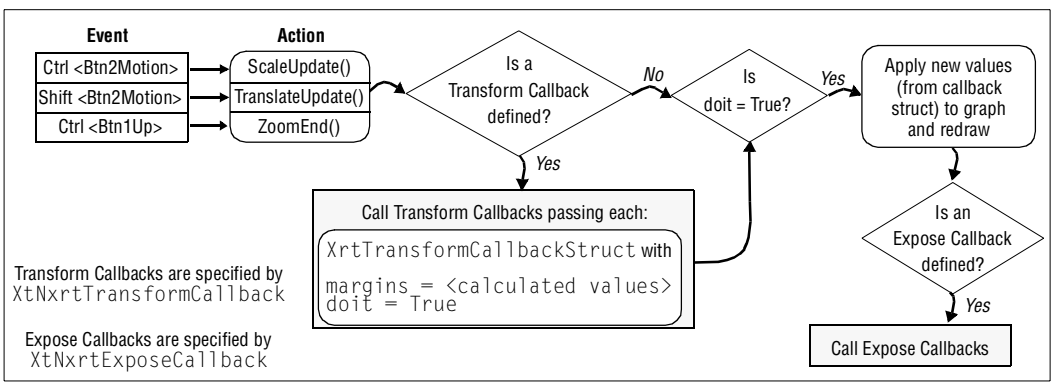


Figure 75 ScaleUpdate(), TranslateUpdate() and ZoomEnd() Action Routines

1. The Graph Margin resources are: `XtNxrtGraphMarginBottom`, `XtNxrtGraphMarginTop`, `XtNxrtGraphMarginLeft`, and `XtNxrtGraphMarginRight`.

Controlling Interaction

You can use `XtNxrtTransformCallback` to control scaling, translation, or zooming. 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
    Boolean     reset;  /* Read-only */
    Position    left_margin;
    Position    right_margin;
    Position    top_margin;
    Position    bottom_margin;
    Boolean     doit;
} XrtTransformCallbackStruct;
```

A Transform callback can change the *left_margin*, *right_margin*, *top_margin*, *bottom_margin*, and *doit* parameters. *ScaleUpdate()*, *TranslateUpdate()*, and *ZoomEnd()* apply these parameters to the graph. For example, to constrain scaling, examine and change the margin parameters in a Transform callback as shown in Figure 76:

```
void
limit(Widget mygraph, XtPointer cd, XtPointer cbs)
{
    XrtTransformCallbackStruct *tcbs = (XrtTransformCallbackStruct *)cbs;

    if (! cbs->reset){
        if (cbs->left_margin   < -100)cbs->left_margin   = -100;
        if (cbs->left_margin   >  100)cbs->left_margin   =  100;
        if (cbs->right_margin  < -100)cbs->right_margin  = -100;
        if (cbs->right_margin  >  100)cbs->right_margin  =  100;
        if (cbs->top_margin    < -100)cbs->top_margin    = -100;
        if (cbs->top_margin    >  100)cbs->top_margin    =  100;
        if (cbs->bottom_margin < -100)cbs->bottom_margin = -100;
        if (cbs->bottom_margin >  100)cbs->bottom_margin =  100;
    }
}
```

Figure 76 Callback procedure that limits Margins to ±100

```
To register the above callback routine, use the following call:
   XtAddCallback(mygraph,
           XtNxrtTransformCallback, limit, NULL);
```

6.4.2 Axis Zooming

The *ZoomAxisEnd()* action zooms into the graph, keeping the axis visible. The zoom area is defined by the "zoom rubberband" (created by the *ZoomAxisUpdate()* action). This routine alters the `XtNxrt[XYY2]AxisMax` and `XtNxrt[XYY2]AxisMin` resources. Figure 77 illustrates this action routine and shows where any ZoomAxis and Expose callbacks you add are called.

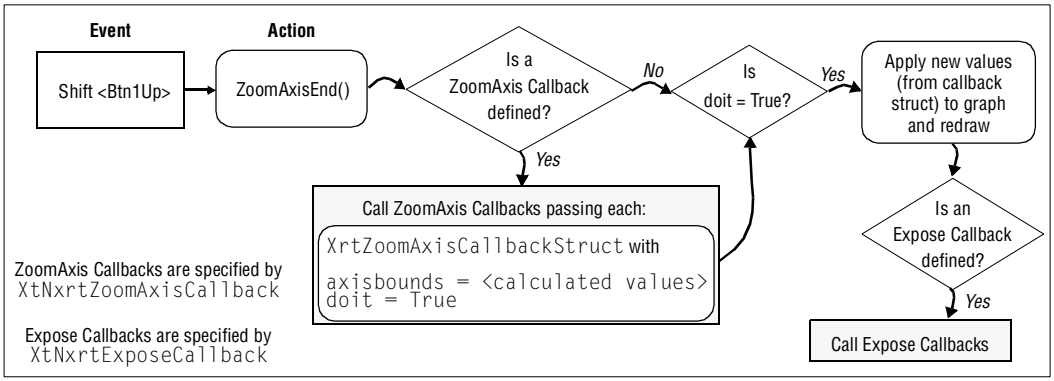


Figure 77 ZoomAxisEnd() Action Routine

Controlling Interaction

You can use `XtNxrtZoomAxisCallback` to control axis zooming. 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
    Boolean     reset;      // Read-only
    float       new_xaxis_min;
    float       new_xaxis_max;
    float       new_yaxis_min;
    float       new_yaxis_max;
    float       new_y2axis_min;
    float       new_y2axis_max;
    Boolean     doit;
} XrtZoomAxisCallbackStruct;
```

A ZoomAxis callback can change the *new_xaxis_min*, *new_xaxis_max*, *new_yaxis_min*, *new_yaxis_max*, *new_y2axis_min*, *new_y2axis_max*, or *doit* parameters. *ZoomAxisEnd()* applies these parameters to the graph. For example, to constrain zooming to certain axis bounds, examine and change the axis parameters in a ZoomAxis callback.

6.4.3 Resetting Scaling, Translation, and Zooming

The *ViewReset()* action resets interactive scaling, translation and zooming by setting Margin and Axis min/max UseDefault resources to `True`. It does not reset these resources unless there has been user-interaction with the widget.

ViewReset() triggers any registered Transform callbacks, passing each a pointer to an `XrtTransformCallbackStruct` with *reset* and *doit* set to `True`. This action then triggers any registered ZoomAxis callbacks, passing each a pointer to an `XrtZoomAxisCallbackStruct` with *reset* and *doit* set to `True`.

An application can deny either the Graph Margin or Axis reset attempt using a Transform or ZoomAxis callback. With either callback, to deny a reset, change the

doit parameter to `False` in the callback when *reset* is `True`. Changes to the other parameters are ignored.

ViewReset() also triggers any registered Expose callbacks.

6.4.4 Rotation

The *RotateUpdate()* action updates interactive rotation of the graph. Only 3D bar, stacking bar and pie charts with `XtNxrtGraphDepth` greater than zero can be interactively rotated. This routine alters the `XtNxrtGraphRotation` and `XtNxrtGraphInclination` resources. Figure 78 illustrates this action routine and shows where any Rotate and Expose callbacks you add are called.

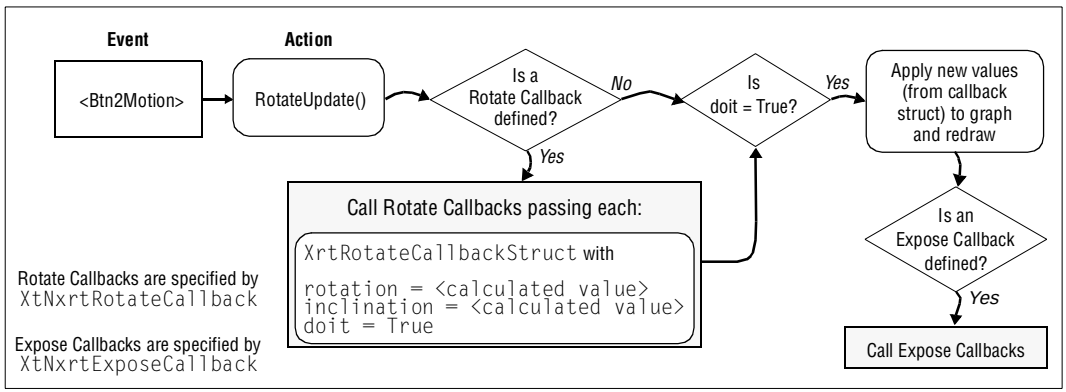


Figure 78 RotateUpdate() Action Routine

Controlling Rotation

You can use `XtNxrtRotateCallback` to control rotation. 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
    int         rotation;
    int         inclination;
    Boolean     doit;
} XrtRotateCallbackStruct;
```

A Rotate callback can change the *rotation*, *inclination*, and *doit* parameters. *RotateUpdate()* applies these parameters to the graph. For example, to constrain inclination between 15 and 30, examine and change the *inclination* parameter in a Rotate callback.

6.4.5 Disabling and Disallowing Interactions

The easiest way to completely disable a particular user interaction is to set its translation to *NoAction()*. See section 6.3 on page 111 for information on disabling *all* interactions. The following code disables all interactions individually:

```
*.XtXrtGraph.translations: #override \n\
    Ctrl          <Btn1Motion>:      NoAction() \n\
    ~Ctrl ~Shift <Btn2Motion>:       NoAction() \n\
    Shift         <Btn2Motion>:      NoAction() \n\
    Ctrl          <Btn2Motion>:      NoAction() \n\
    <Key>r:                          NoAction()
```

You can use a callback to disallow an interaction update on a case-by-case basis. Set the *doit* parameter of the appropriate interaction's passed callback structure to `False` in your callback. The action routine that triggered the callback checks this parameter, and does not apply the changes if *doit* is `False` (as shown by Figure 74 through Figure 78). The Modify callbacks triggered by the *ModifyEnd()* action are the one exception; this callback does nothing except notify the application that a user interaction has finished.

The following callback procedure disallows a user's attempt to reset the graph by setting the `XrtTransformCallbackStruct`'s *doit* parameter to `False`:

```
void
deny_reset(Widget w, XtPointer cd, XtPointer cbs)
{
    XrtTransformCallbackStruct *tcbs =
        (XrtTransformCallbackStruct *)cbs;

    if (cbs->reset){
        cbs->doit = False;
    }
}
```

To register the above callback routine, use the following call:

```
XtAddCallback(graph,
    XtNxrtTransformCallback, deny_reset, NULL);
```

6.5 Ending User Interaction

The *ModifyEnd()* action is called to notify the application that a user interaction has finished. An application can receive this notification by using `XtNxrtModifyCallback`. Each callback procedure is passed a pointer to an `XrtModifyCallbackStruct` structure with the *reason* parameter set to `XRT_REASON_MODIFY_END`. Changing the parameters in the structure has no effect.

6.6 Interacting with Graph Data

XRT/graph provides two different ways to handle user interaction with graph data: through action routines/callbacks (section 6.6.1), or directly through event handling procedures (section 6.6.2). Using the action routines is generally easier to program.

6.6.1 Action Routines and Callbacks

You can use a translation table to notify your application when a user attempts to interact with the graph data and receive either the graph coordinates clicked, or the closest data values to the user event. XRT/graph provides two action routines: *Pick()* and *Map()*. To use these routines, you must do the following:

- Install a translation to map a user event to the appropriate action routine (since neither routine has a default translation).
- Register Pick or Map callbacks (described below). These are triggered during the *Map()* or *Pick()* actions and are passed the result of the *XrtPick()* and *XrtMap()* procedures.

Section E.2 on page 252 fully describes XRT/graph's action routines.

Pick() Action

The *Pick()* action determines the graph data values closest to the location of a user event and passes them to a callback procedure. It uses the *XrtPick()* procedure. You can use the *Pick()* action to find out which set and point a user has clicked nearest. For more information on *XrtPick()*, see page 120. Figure 79 illustrates this action routine and shows where Pick callbacks are invoked.

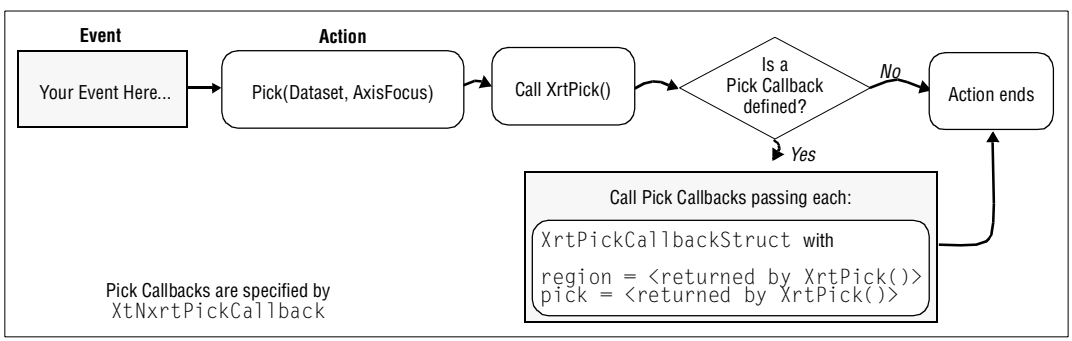


Figure 79 Pick() Action Routine

The *Pick()* action must be assigned to an event in order for it to be called, for example:

```
*.XtXrtGraph.translations: #override \n\
    ~Shift ~Ctrl <Btn1Down>:  Pick()
```

Pick() takes two optional parameters that specify the dataset and axis focus to use in the call to *XrtPick()*. By default, *Pick()* uses all datasets and both axes.

You must use `XtNxrtPickCallback` to register a callback to receive the results of the *XrtPick()* 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 */
    XrtRegion       region; /* Read-only */
    XrtPickResult   pick;   /* Read-only */
} XrtPickCallbackStruct;
```

Changing any of the fields in this structure has no effect.

Map() Action

The *Map()* action determines the graph coordinates that map to the pixel coordinates of a user event and passes them to a callback procedure. It uses the *XrtMap()* procedure. You can use the *Map()* action to find out which (x,y) value a user has clicked on. For more information on *XrtMap()*, see page 123. The way *Map()* works is similar to *Pick()*, illustrated in Figure 79.

The *Map()* action must be assigned to an event in order for it to be called, for example:

```
*.XtXrtGraph.translations: #override \n\
    ~Shift ~Ctrl <Btn1Down>:  Map()
```

Map() takes one optional parameter that specifies which Y-axis to use for the coordinates in the call to *XrtMap()*. By default, *Map()* uses the first Y-axis.

You must use `XtNxrtMapCallback` to register a callback to receive the results of the *XrtMap()* 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 */
    XrtRegion       region; /* Read-only */
    XrtMapResult    map;    /* Read-only */
} XrtMapCallbackStruct;
```

Changing any of the fields in this structure has no effect.

Example

Figure 80 provides an example of an action routine that replaces XRT/graph's default zoom translation. It uses *XrtMap()* to control axis bounds.

```
int x1, y1, x2, y2;

static void
start_zoom(Widget w, XEvent *event)
{
    XButtonEvent *be = (XButtonEvent *) event;

    x1 = be->x;
    y1 = be->y;
}

static void
end_zoom(Widget w, XEvent *event)
{
    XrtMapResult min, max;
    int     top, bottom, left, right;
    XButtonEvent *be = (XButtonEvent *) event;

    x2 = be->x;
    y2 = be->y;

    left   = MIN(x1, x2)
    right  = MAX(x1, x2);
    top    = MIN(y1, y2);
    bottom = MAX(y1, y2);

    XrtMap(w, 1, left, bottom, &min);
    XrtMap(w, 1, right, top, &max);

    XtVaSetValues(w,
        XtNxrtXAxisMin,XrtFloatToArgVal(min.x),
        XtNxrtYAxisMin,XrtFloatToArgVal(min.y),
        XtNxrtXAxisMax,XrtFloatToArgVal(max.x),
        XtNxrtYAxisMax,XrtFloatToArgVal(max.y),
        0);
}

static char translations[] = "#override   \n\
     Ctrl<Btn1Down>:    ModifyStart(Zoom) start_zoom()      \n\
     Ctrl<Btn1Motion>:  ZoomUpdate()      \n\
     Ctrl<Btn1Up>:    end_zoom() ModifyEnd(Zoom) ";
```

Figure 80 Action Routine to replace default Zoom translation

6.6.2 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/graph program can make use of the event handling mechanisms to react to events that happen over a graph widget.

The following code will register the procedure *my_event_proc()* with the Xt Intrinsics. Whenever a mouse button event occurs over the graph window, *my_event_proc()* will be called.

```
Widget      my_graph;
XtAddEventHandler( my_graph,
    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.

Within the XEvent structure are the pixel coordinates of the event. XRT/graph provides procedures for mapping the pixel coordinates of an event to:

- set and point indices of the data displayed closest to the event coordinates.
- graph coordinates.

XrtPick()

The *XrtPick()* procedure provides an easy mechanism for the programmer to discover "what the user is pointing at" when he or she clicks the mouse on a graph.

XrtPick() takes a graph handle, dataset specification, (x,y) pixel coordinates, a pointer to an `XrtPickResult` structure and a focus hint as arguments. It fills in the `XrtPickResult` structure with information about the data values closest to the specified pixel coordinates on the specified graph.

```
XrtRegion
XrtPick(
    Widget          graph,
    XrtDsGroup      ds_group,
    int             pix_x,
    int             pix_y,
    XrtPickResult   *pick,
    XrtFocus        focus
    )
```

ds_group indicates which dataset the pick results will apply to. In combination graphs it can be set to `XRT_DATASET1` or `XRT_DATASET2` to restrict the pick results to one of the datasets. If set to `XRT_DATASET1 | XRT_DATASET2`, the closest point from either of the datasets will be returned. When the graph is not a combination graph, set this to `XRT_DATASET1`.

focus can be XRT_XFOCUS or XRT_YFOCUS to constrain the search for the closest point in the direction of the X- or Y-axis. To search for the closest point in cartesian space, set *focus* to XRT_XFOCUS | XRT_YFOCUS.

XrtPick() returns one of the values described below:

XRT_RGN_NOWHERE	The given pixel coordinates are not close enough to anything to be picked. In this case the XrtPickResult structure is not filled in.
XRT_RGN_IN_GRAPH	The given pixel coordinates are close to one of the displayed data values in the graph. In this case, the set and point values in the XrtPickResult structure identify the closest data. distance indicates how close the picked data is to the pixel coordinates.
XRT_RGN_IN_LEGEND	The given pixel coordinates are close to one of the elements in the legend area. In this case, set will indicate which set is closest (point will be XRT_RGN_IN_LEGEND). distance indicates how close the picked data is to the pixel coordinates.
XRT_RGN_IN_HEADER	The given pixel coordinates are not in the graph, but are in the header area. The XrtPickResult structure is not filled in unless the header overlaps the data area.
XRT_RGN_IN_FOOTER	The given pixel coordinates are not in the graph, but are in the footer area. The XrtPickResult structure is not filled in unless the footer overlaps the data area.

The XrtPickResult structure is defined as:

```
typedef struct {
    int         pix_x, pix_y;
    int         dataset;
    int         set, point;
    int         distance;
} XrtPickResult;
```

The fields are broken down as follows:

pix_x	The x pixel coordinate passed to *XrtPick()*.
pix_y	The y pixel coordinate passed to *XrtPick()*.
dataset	Either XRT_DATASET1 or XRT_DATASET2 to indicate which dataset the results apply to.
set	The logical set index of the data element displayed closest to the pixel coordinates. This can be XRT_RGN_NOWHERE if no data is displayed.

`point` The point index of the data element displayed closest to the pixel coordinates. This can be `XRT_RGN_NOWHERE` if no data is displayed.

`distance` The screen distance (in pixels) between the given pixel coordinates and the on-screen display of the data indexed by `set` and `point`, as determined by focus.

Things are slightly more complex when the picked element corresponds to the *other* slice of a pie. In this case, `set` is `XRT_OTHER_SLICE` and `point` indicates the selected point. The example event procedure in Figure 81 determines the bar that is closest to the point clicked and puts its value in the graph's header.

XrtUnpick()

The *XrtUnpick()* procedure is the opposite of *XrtPick()*. It determines the pixel coordinate of a data point and set. It is provided for advanced X application developers who use Xlib routines to draw on top of graphs. See Appendix D on page 211 for more details.

```
static void
Select(Widget graph, XEvent *event)
{
    int     idx;
    char    *nhs[2], buffer[100];
    XButtonEvent    *button_event;

    XrtPickResult   p;
    int     rc;
    XrtDataHandle tmp_data;

    button_event = (XButtonEvent *) event;

    rc = XrtPick(graph, XRT_DATASET1,
            button_event->x, button_event->y,
            &p, XRT_XFOCUS | XRT_YFOCUS);
    if (rc == XRT_RGN_IN_GRAPH) {
        XtVaGetValues(graph,
            XtNxrtData, &tmp_data, XmNuserData, &idx, NULL);
        sprintf(buffer, "%s -- Day %s: $%.2f",
            header_strings[idx][0],
            tdays_0990[p.point],
            XrtDataGetYElement(tmp_data, p.set, p.point) );
        nhs[0] = buffer;
        nhs[1] = NULL;
        XtVaSetValues(graph, XtNxrtHeaderStrings, nhs, NULL);
    }
}
```

Figure 81 Using XrtPick() in an Event Procedure

XrtMap()

The *XrtMap()* procedure maps from pixel coordinates to graph coordinates (i.e. it maps from pixel space to graph space). It takes a graph handle, an axis specification, (x,y) pixel coordinates and a pointer to a map structure and writes the pixel coordinates, axis specification and the floating-point graph coordinates into the map structure.

Calling *XrtMap()* with `yaxis` set to 1 maps to the graph coordinates defined by the first Y-axis. Setting `yaxis` to 2 maps to the graph coordinates defined by the second Y-axis.

```
XrtRegion
XrtMap(
    Widget          graph,
    int             yaxis,
    int             pix_x,
    int             pix_y,
    XrtMapResult    *map
    )
```

XrtMap() returns `XRT_RGN_NOWHERE`, `XRT_RGN_IN_FOOTER`, `XRT_RGN_IN_HEADER` or `XRT_RGN_IN_GRAPH`.

The `XrtMapResult` structure is defined as follows:

```
typedef struct {
    int pix_x, pix_y;
    int yaxis;
    float x, y;
} XrtMapResult;
```

The fields are broken down as follows:

`pix_x` The x coordinate passed into *XrtMap()*.

`pix_y` The y coordinate passed into *XrtMap()*.

`yaxis` The axis specification passed into *XrtMap()*.

`x` The mapped graph coordinate X-value. If the chart type is pie, `x` is `XRT_HUGE_VAL`.

`y` The mapped graph coordinate Y-value. If the chart type is pie, `y` is `XRT_HUGE_VAL`.

The example event procedure in Figure 82 captures the graph coordinates of a left mouse down and a left mouse up event. It then zooms the plot in on the rectangular area specified. It assumes the existence of *fmin()* and *fmax()*, which return the floating point minimum or maximum of their arguments.

```
void
zoom_plot(graph, client_data, event)
Widget  graph;
caddr_t client_data;
XEvent  *event;
{
    XButtonEvent*button_event;
    static float x1, y1;
    XrtMapResult m;
    int         rc;

    if (event->type != ButtonPress && event->type != ButtonRelease) {
        return();
    }
    button_event = (XButtonEvent *) event;

    if ( button_event->button == Button1 &&
        XrtMap(graph, 1, button_event->x,
            button_event->y, &m) == XRT_RGN_IN_GRAPH)

        if ( event->type == ButtonPress ) {
            x1 = m.x;
            y1 = m.y;
        }
        else { /* Is a button release */
            XtVaSetValues(my_plot,
                XtNxrtXAxisMin,XrtFloatToArgVal( fmin(x1, m.x) ),
                XtNxrtXAxisMax,XrtFloatToArgVal( fmax(x1, m.x) ),
                XtNxrtYAxisMin,XrtFloatToArgVal( fmin(y1, m.y) ),
                XtNxrtYAxisMax,XrtFloatToArgVal( fmax(y1, m.y) ),
                NULL);
        }
}
```

Figure 82 A "zooming" Event Procedure

XrtUnmap()

The *XrtUnmap()* procedure is the opposite of *XrtMap()*. It maps from graph coordinates to pixel coordinates. It is available for advanced X application developers who use Xlib routines to draw on top of graphs. See Appendix D on page 211 for more details.

6.7 Window Resizing

Most applications should allow the user to resize a window containing a graph widget, and have the widget adjust to the new window size.

The graph widget can be located inside any of the Motif Composite widgets. Usually the Composite widget will determine the appropriate size for the graph widget, and the graph widget will adjust automatically.

Expose & Resize Callbacks

An application can find out when the graph widget is exposed or resized by specifying a callback function using the `XtNxrtExposeCallback` or `XtNxrtResizeCallback` 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 `XrtCallbackStruct` which contains additional information about the callback. See Appendix A on page 171 for complete details. The resize callback is called after the graph is resized. The expose callback is called after the graph is redrawn.

An application can use these callbacks to draw on top of the graph image using Xlib functions. Another use is to adjust the widget resources depending on the size of the XRT/graph widget.

The following code registers a resize callback function:

```
XtAddCallback(mygraph, XtNxrtResizeCallback,
    resize_proc, &my_data);
```

The following resize callback causes the legend to disappear when the graph widget gets too small:

```
void
resize_proc(w, client_data, call_data)
Widget          w;
caddr_t         client_data;
XrtCallbackStruct *call_data;
{
    Dimension width;

    XtVaGetValues(w,
        XtNwidth, &width,
        NULL);

    if ( width <= 300 )
        XtVaSetValues(w,
            XtNxrtLegendShow, False,
            NULL);
    else
        XtVaSetValues(w,
            XtNxrtLegendShow, True,
            NULL);
}
```

7

Advanced Programming Topics

Adding a Second Y-axis ■ *Combination Graphs*
Adding Text Objects ■ *Batching Resource Updates*
Internationalization Support ■ *Fast Update Procedures*
Porting XRT/graph to Microsoft Windows ■ *Customizing Warning Messages*
Using C++ with XRT/graph ■ *Using UIL with XRT/graph*
Converting Resource Values to Strings

7.1 Adding a Second Y-axis

The leftmost Y-axis is called the *first Y-axis* or *primary Y-axis* in XRT/graph. It is possible to add a second Y-axis (Y2) to the right side of the graph. If the graph is inverted, the first Y-axis is at the bottom, and the second Y-axis is at the top. If the X-axis is reversed the position of the Y-axes is switched.

There are two ways to create a second Y-axis on a graph. The simplest way is to specify a relationship between the first and second Y-axis. A second way is to create a combination graph by attaching a second dataset. Combination graphs are discussed in section 7.2 on page 128.

Note that when the second dataset is the *only* dataset in the graph, the Y2-axis becomes the primary Y-axis, and its resources are used.

Specifying a Relationship

A relationship between the Y-axes is specified using the `XtNxrtYAxisMult` and `XtNxrtYAxisConst` resources. XRT/graph will create the second Y-axis from the first Y-axis using the equation:

```
Y2 = (Y1 * XtNxrtYAxisMult) + XtNxrtYAxisConst
```

For example, if a graph is plotting temperatures that have been measured in degrees Celsius, the following code would create a second Y-axis for Fahrenheit:

```
XtVaSetValues(graph,
    XtNxrtYAxisMult,
            XrtFloatToArgVal(9.0/5.0),
    XtNxrtYAxisConst, XrtFloatToArgVal(32.0),
    XtNxrtY2Precision,0,
    XtNxrtY2Title, "Fahrenheit",
    NULL);
```

To eliminate the second Y-axis, disable the relationship by setting `XtNxrtYAxisMult` to 0.0.

`XtNxrtY2AxisLogarithmic` is ignored if a relationship is specified between the Y-axes.

Precision Issues

When `XtNxrtY2PrecisionUseDefault` is `True`, XRT/graph handles all Y2-axis precision issues. However, if you are explicitly setting Y2-axis precision, it is important to allow enough digits. Setting the precision too small may cause unexpected behavior, such as the disappearance of Y2 annotation. In general, the Y2 precision should be at least as many digits as the maximum of:

- the number of digits of Y1 precision plus the implied precision of `XtNxrtYAxisMult`.
- the implied precision of `XtNxrtYAxisConst`.

Since IEEE floating point arithmetic is 6-digit precision, it is important that neither Y-axis require more than 6 digits of precision.

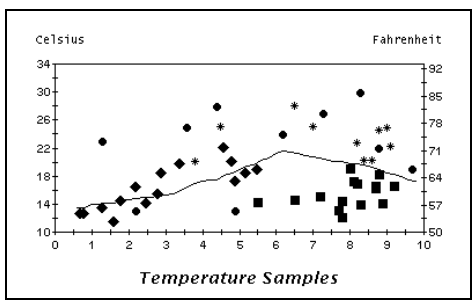


Figure 83 Second Y-axis Showing a Linear Relationship

7.2 Combination Graphs

Combination graphs have two datasets displayed in the same graph area. Plot, area, bar, stacking bar, HiLo, HiLoOpenClose, bubble and candle charts can all be used in combination graphs.

A combination graph has a second dataset attached to it using the XtNxrtData2 resource. The second graph type is set using XtNxrtType2 (defaults to XRT_TYPE_BAR). By default, the first graph will display in front (i.e. closer to the viewer). This can be switched by setting XtNxrtFrontDataset to XRT_DATASET2. Markers can also be used with combination graphs. The XtNxrtMarkerDataset resource determines which dataset is used for marker positioning.

Special Rules

There are several special rules that apply just to combination graphs:

1. If either graph is a pie, polar, radar, or filled radar chart, the second graph will not be displayed.
2. When the X-axis annotation method is Point-labels, only the *first* graph's Point-labels (specified by XtNxrtPointLabels) are used.

Second Y-axis

The minimum, maximum, numbering increment, ticking increment and precision of the second graph is programmable using the Y2 versions of the axis resources. The second Y-axis can be logarithmic if no axis relationship has been specified.

Second Data Style

Data styles for the second graph are programmed using the XtNxrtDataStyles2 and XtNxrtDataStyles2UseDefault resources.

Legend

The legend in a combination graph is a concatenation of each graph's Set-labels. Point-labels and Set-labels for the second graph are programmed using XtNxrtPointLabels2 and XtNxrtSetLabels2. NULL labels or zero-length strings are not included in the legend.

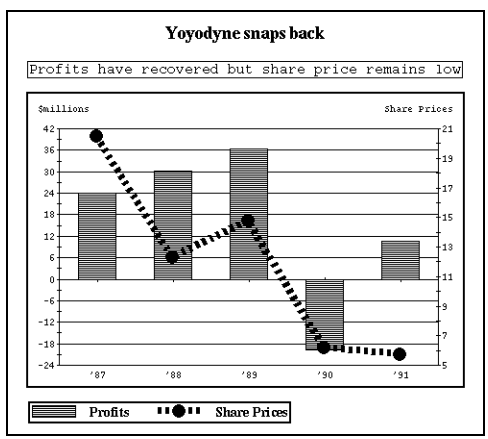


Figure 84 Combination Plot and Bar Graph

7.3 Adding Text Objects

Any type of graph can display one or more *text objects*. A text object is an independent rectangular region drawn over the graphed data. They can be attached to the graph in one of four ways:

- To a pixel location on the screen,
- To graph data X- and Y-value,
- To a data point (set, point index), or
- Above or below a data point (set, point index, Y-value).

An application can use text objects to highlight special points on the graph, or as a more general-purpose label. Any number of text objects can be attached to a graph, dynamically created, updated, and destroyed.

Text objects are referenced by a handle of type `XrtTextHandle`. A collection of methods create, set attributes, get attributes, and destroy text objects.

See the *text_area.c* demo program in *$XRTHOME/src/graph/demos/text_area* for example code.

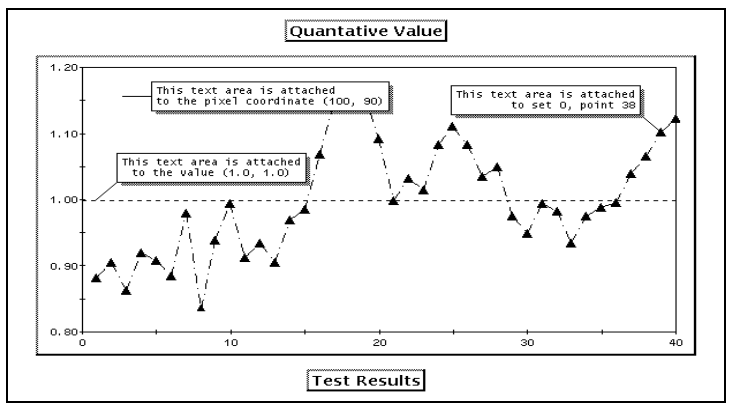


Figure 85 Text Objects attached to pixel, graph coordinates, and data point

Creating a Text Object

To add a text object to a graph, create it as a child of the graph, as shown by the following example:

```
XrtTextHandle    text_obj;
char *strings[] = {"Hello World!", NULL};

text_obj = XrtTextAreaCreate(graph);
XrtTextAreaSetValues(text_obj, XRT_TEXT_STRINGS, strings, NULL);
```

Setting and Getting Attributes

Each text object has attributes that control its position and appearance. The *XrtTextAreaSetValues()* procedure sets the value of one or more attributes, like

XtVaSetValues(). The *XrtTextAreaGetValues()* procedure retrieves the current value of one or more attributes, like *XtVaGetValues()*.

The previous code example used *XrtTextAreaSetValues()* to set the text of the text object. Figure 86 summarizes the text object attributes; for full details, see Appendix B on page 205.

Positioning Attributes

XRT_TEXT_ANCHOR	XRT_TEXT_ATTACH_DATASET	XRT_TEXT_ATTACH_PIXEL_[XY]
XRT_TEXT_ATTACH_POINT	XRT_TEXT_ATTACH_SET	XRT_TEXT_ATTACH_TYPE
XRT_TEXT_ATTACH_VALUE_[XY]	XRT_TEXT_OFFSET	XRT_TEXT_HEIGHT
XRT_TEXT_WIDTH	XRT_TEXT_X	XRT_TEXT_Y

Appearance Attributes

XRT_TEXT_ADJUST	XRT_TEXT_BACKGROUND_COLOR	XRT_TEXT_BORDER
XRT_TEXT_BORDER_WIDTH	XRT_TEXT_FONT	XRT_TEXT_FOREGROUND_COLOR
XRT_TEXT_IS_CONNECTED	XRT_TEXT_IS_SHOWING	XRT_TEXT_ROTATION
XRT_TEXT_STRINGS	XRT_TEXT_IMAGE	XRT_TEXT_IMAGE_MINIMUM_SIZE
XRT_TEXT_IMAGE_LAYOUT		

Miscellaneous Attributes

XRT_TEXT_NAME	XRT_TEXT_PRINT_FONT

Figure 86 Text Object Attribute Summary

Text

The text displayed in a text object is specified with the XRT_TEXT_STRINGS attribute. Text objects can display multiple lines of text.

The alignment of multi-line text is specified with the XRT_TEXT_ADJUST attribute. Text can be left-aligned (XRT_ADJUST_LEFT), centered (XRT_ADJUST_CENTER), or right-aligned (XRT_ADJUST_RIGHT).

Attachment Point and Positioning

Use the XRT_TEXT_ATTACH_TYPE attribute to specify how a text object is attached to the graph. It can be attached to pixel coordinates (XRT_TEXT_ATTACH_PIXEL), data X- and Y- value (XRT_TEXT_ATTACH_VALUE), a data point (XRT_TEXT_ATTACH_DATA), or to a Y-value above or below a data point (XRT_TEXT_ATTACH_DATA_VALUE). The type chosen determines which other attributes are used for the actual attachment point:

Attachment Type	Attributes that Specify Attachment Point
Pixel coordinates	XRT_TEXT_ATTACH_PIXEL_X XRT_TEXT_ATTACH_PIXEL_Y
Data X- and Y-value	XRT_TEXT_ATTACH_DATASET XRT_TEXT_ATTACH_VALUE_X XRT_TEXT_ATTACH_VALUE_Y

Attachment Type	Attributes that Specify Attachment Point
Data Point	XRT_TEXT_ATTACH_DATASET XRT_TEXT_ATTACH_POINT XRT_TEXT_ATTACH_SET
Data Point at a Y-value	XRT_TEXT_ATTACH_DATASET XRT_TEXT_ATTACH_POINT XRT_TEXT_ATTACH_SET XRT_TEXT_ATTACH_VALUE_Y

Once you have determined an attachment point, you can position the text object using the XRT_TEXT_ANCHOR and XRT_TEXT_OFFSET attributes. Text objects can be anchored at one of eight compass points, centered over the attachment point, or you can allow XRT/graph to determine the best position.

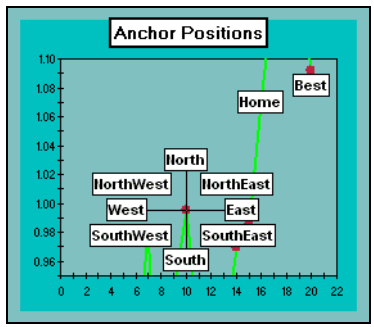


Figure 87 Anchoring positions using custom offsets

The following example attaches and positions a text object:

```
XrtTextAreaSetValues(my_text,
    XRT_TEXT_ATTACH_TYPE,       XRT_TEXT_ATTACH_DATA,
    XRT_TEXT_ATTACH_POINT,      3,
    XRT_TEXT_ATTACH_SET,        2,
    XRT_TEXT_ANCHOR,            XRT_ANCHOR_NORTH,
    XRT_TEXT_OFFSET,            10,
    NULL);
```

Connecting Line

You can draw a line between a text object and its attachment point using the XRT_TEXT_IS_CONNECTED attribute. This attribute should be used with the XRT_TEXT_OFFSET attribute to set a distance between the text object and the attachment point.

Font

When a text object is created, it uses the same font as XtNxrtAxisFont. Use the XRT_TEXT_FONT attribute to change the font. This attribute is specified the same as header, footer, and legend fonts. See section 4.8 on page 72 for further details.

Border

Text objects can be enhanced with a border using the XRT_TEXT_BORDER and XRT_TEXT_BORDER_WIDTH attributes. Any valid XRT/graph border type may be used; see section 4.16 on page 91 for details on borders.

Colors

You can set a background and foreground color for a text object using XRT_TEXT_BACKGROUND_COLOR and XRT_TEXT_FOREGROUND_COLOR. For more details on specifying colors, see section 2.8 on page 29.

Images

You can set a background image for a text object by using the XRT_TEXT_IMAGE, XRT_TEXT_IMAGE_LAYOUT, and XRT_TEXT_IMAGE_MINIMUM_SIZE attributes. For more information, see Background Images on page 91.

Text Rotation

The text contained in a text object can be rotated using the XRT_TEXT_ROTATION attribute. You can select 90-degree (XRT_ROTATE_90), 270-degree (XRT_ROTATE_270), or no rotation (XRT_ROTATE_NONE).

Text Object Performance

Applications should batch the creation or manipulation of multiple text objects. See section 7.4 on page 133 for more information on batching.

Removing a Text Object

To remove a text object from the display, set XRT_TEXT_IS_SHOWING to False. To permanently destroy a text object, call *XrtTextAreaDestroy()*.

7.4 Batching Resource Updates

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

It is recommended that you batch around the creation or updating of multiple text areas.

7.5 Internationalization Support

Locale

XRT/graph allows you to make use of ANSI and POSIX locale standards to improve the international usability of your applications. In particular, the *LC_TIME* environment variable is used by the time-axis formatting function (See Figure 88 below). Axis annotation also makes use of the *LC_NUMERIC* locale. For example,

some locales use a comma instead of a period for the decimal point. See your system documentation on *setlocale()* for information on setting the locale for your application.

The XRT/graph I/O routines such as *XrtDataCreateFromFile()* and *XrtSaveDataToFile()* use the "C" locale for numeric values.

X Window Fonts

Any X Window font supported on your Server may be used for the graph header, footer, legend or graph areas, as well as for individual text areas.

ISO Latin1 Fonts

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.

The printing procedure *XrtDrawPS()* generates PostScript that uses the full ISO Latin1 character set if it is supported on the PostScript imaging device.

2-byte Fonts

2-byte fonts such as Kanji may be specified for the header, footer and legend areas, as well as for individual text areas.

With the exception of Japanese characters, the *XrtDrawPS()* procedure cannot print text that uses 2-byte fonts. Use an alternative printing procedure, such as *XrtOutputPS()* or *XrtOutputXwd()* to print graphs containing 2-byte fonts.

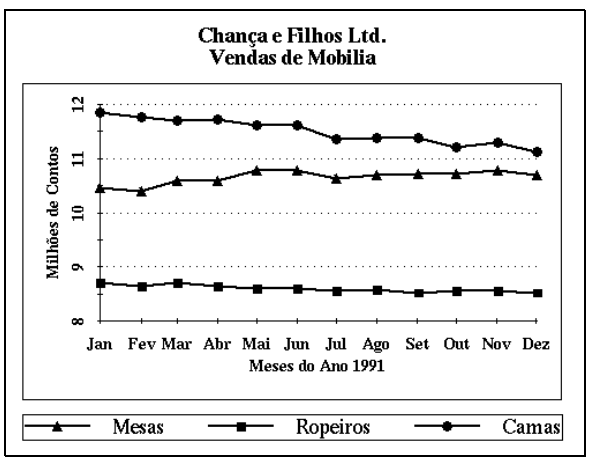


Figure 88 Time-axis with LANG Set to "portuguese"

7.6 Fast Update Procedures

Normally, when adding new data to a graph, you simply update the graph's data object(s) and reset the `XtNxrtData` and/or `XtNxrtData2` resources to the respective `XrtDataHandle`. Whenever this happens, XRT/graph examines the entire data and repaints the entire graph area.

Note: the Fast Update procedure is not available for use with bar, pie, radar, filled radar, and polar graph types.

The Fast Update procedures provide an alternative and potentially much faster way to add new data to existing graphs. The fast update procedures will only be faster if the new data falls within the graph's current axes bounds. If any of the new data falls outside these bounds, the entire graph will be repainted.[1]

The Fast Update procedures are fully documented in Appendix D on page 211. The following is a brief description of the procedures:

XrtArrDataFastUpdate()	This procedure graphs new points that have been added to each set in an array dataset.
XrtGenDataFastUpdate()	This procedure graphs new points that have been added to one set in a general dataset.
XrtArrCheckAxisBounds()	This procedure examines new points that have been added to each set in an array dataset, and determines which (if any) axes bounds would be exceeded if these new points were displayed.
XrtGenCheckAxisBounds()	This procedure examines new points that have been added to one set in a general dataset, and determines which (if any) axes bounds would be exceeded if these new points were displayed.
XrtArrDataShiftPts()	This procedure copies existing points (in all sets) to the left. Use when axis bounds have been exceeded. Shifted-out points are set to the hole value.
XrtGenDataShiftPts()	This procedure copies existing points (in one set) to the left. Use when axis bounds have been exceeded. Shifted-out points are set to the hole value.

To use the Fast Update procedures you should:

- Add the new points to the graph's existing data object referenced by the `XtNxrtData` or `XtNxrtData2` resource.
- Optionally call the appropriate CheckAxisBounds procedure if you want to know in advance if the update will be fast or regular speed.

1. Fast Update procedures trigger expose callbacks only when new data falls outside the axes bounds.

- Call the appropriate Fast Update procedure to perform the update.

Any non-solid line patterns (like dashed or dotted) are not guaranteed to be continued correctly using Fast Update. You should use solid lines if this is important to your application.

The following code sample shows how a scrolling strip chart is implemented using the Fast Update procedures:

```
1 if (XrtArrDataAppendPts(data, new_xvalue, new_yvector)) {
2   bound = XrtArrCheckAxisBounds(graph, 1, 1); /* new points fit in axes? */
3       if ((bound & XRT_GTX) == 0) {          /* less than xmax? */
4           XrtArrDataFastUpdate(graph, 1, 1); /* Fast Update */
5       }
6       else {
7           new_numpoints = XrtDataGetNPoints(data, 0) - increment;
8           /* shift X values */
9           XrtArrDataShiftPts(data, 0, increment, new_numpoints);
10          XrtDataSetNPoints(data, 0, new_numpoints);
11          xmin = XrtDataGetXElement(data, 0, 0);
12          xmax = XrtDataGetXElement(data, 0, new_numpoints-1) +
13              increment * x_increment;
14          XtVaSetValues(graph,                /* repaint with new axis */
15              XtNxrtData, data,
16              XtNxrtXMin, XrtFloatToArgVal(xmin),
17              XtNxrtXMax, XrtFloatToArgVal(xmax),
18              NULL);
19      }
20 }
```

Figure 89 Scrolling Strip Chart Code

On line 1, new data points are added to the graph's data object (the handle of which must be the current value of the graph's `XtNxrtData` resource).

Line 2 checks to see if the new points fit within the current axes bounds. If they do, the Fast update procedure is called (line 4). Otherwise, the entire dataset is shifted downward (lines 7-13), and then graphed (lines 14-18).

7.7 Porting XRT/graph to Microsoft Windows

XRT/graph provides support for porting Motif XRT/graph applications to Microsoft Windows, using KL Group's Olectra Chart product for the Windows application. With Olectra Chart, your graph application can use the native Windows toolkit and be deployed on Windows 3.1, Windows 95, or Windows NT 3.51 and later. This section describes XRT/graph's cross-platform API and provides tips on dealing with the inherent differences between the two platforms.

It is possible to maintain a single source tree for an application. More complex applications may consider copying their Motif source to Windows to port, and maintaining separate Motif and Windows source trees.

Overview

The Windows toolkit contains pre-defined controls similar to Motif. Olectra Chart's 16- and 32-bit "2D Chart" DLL custom control adds a charting "widget" to the Windows toolkit. This control has the same resources (called properties on Windows), data types, and methods as XRT/graph.[1] Using this common API, it is possible to recompile XRT/graph applications to run on the Microsoft Windows with minimal effort. Applications can use `#ifdef` statements to handle Motif- or Windows-specific code.

Using C++ wrappers can make porting easier since application code is abstracted and less platform-specific. Both XRT/graph and Olectra Chart's 2D Chart control include C++ wrappers.

Creating a Graph Control

Controls are created differently in Windows programs. While a Motif widget is instantiated as a child of another widget with its size and location defined on creation, Windows programs generally define controls in a resource or *.RC* file, without any kind of attachment to other controls.

The class name used when defining a 2D Chart control is "OlectraChart2D" (32-bit DLL) or "OlectraChart2D16" (16-bit DLL).

Once defined, a Windows program can create the control during its initialization, similar to a Motif program. Instead of an *XtCreateManagedWidget()* call that creates an XRT/graph widget as a child of a toplevel shell, Olectra Chart programs use code similar to the following:

```
static HXRT2D graph;
static HWND hwndChart;

hwndChart = GetDlgItem(hWnd, IDCHART); /* defined in .RC file */
graph = XrtCreate();
XrtAttachWindow(graph, hwndChart);   /* joins chart and window /*
```

Instead of the single `Widget` handle, Windows programs need a handle for the 2D Chart control (`HXRT2D`) and one for its window on the screen (`HWND`).

Graph Data

One major difference between XRT/graph and Olectra Chart's 2D Chart control is the data type of graph data. XRT/graph uses `float` but 2D Chart uses `double`. The simplest way to handle this is to define a constant for the data type for both Motif and Windows, for example:

```
#ifdef WIN_DLL
#define XrtFloat double
#endif
#ifdef MOTIF
#define XrtFloat float
#endif
```

1. Although this control is also provided as an OCX, only the DLL shares XRT/graph's cross-platform API.

Setting and Getting Resources

XRT/graph shares with Olectra Chart an API for setting and getting resource values. *XrtSetValues()* works like *XtVaSetValues()*, allowing you to set one or more resources. *XrtGetValues()* works like *XtVaGetValues()* to retrieve the current value of resources. Existing Motif code that called *XtVaSetValues()* to set graph resources can be modified to call *XrtSetValues()* instead. Note that with *XrtSetValues()* you should not call *XrtFloatToArgVal()* or *XrtArgValToFloat()* before accessing resources of type `float`.

Each XRT/graph resource has a corresponding cross-platform name that can be used in both Motif and 2D Chart code on Windows. Figure 90 lists examples for several resources. For a complete reference, see page 267.

```
XRT_AXIS_FONT          XRT_BACKGROUND_COLOR   XRT_DATA            XRT_DATA_STYLES
XRT_FOOTER_BORDER      XRT_FOOTER_STRINGS     XRT_GRAPH_HEIGHT    XRT_HEADER_FONT
XRT_LEGEND_SHOW        XRT_POINT_LABELS       XRT_SET_LABELS      XRT_TYPE
XRT_XANNOTATION_METHODXRT_YAXIS_MAX           XRT_YLABELS         XRT_Y2TITLE
```

Figure 90 Some cross-platform resource names

The following example sets three graph properties, assuming the `my_data` and `labels` variables are defined correctly:

```
XrtSetValues(graph,
    XRT_DATA,              my_data,
    XRT_TYPE,              XRT_TYPE_STACKING_BAR,
    XRT_SET_LABELS,        labels,
    NULL);
```

Resource Setting Using Strings

The *XrtSetPropString()* and *XrtGetPropString()* procedures provide another way to make graph applications more portable. These procedures set/get a resouce value as a string pointer. This is useful for setting colors without recoding to use the Windows `COLORREF` color type. The format of the value is the same as XRT/graph resources in a resource file (see Appendix H on page 277 for resource file syntax). The following example sets the header background color:

```
XrtSetPropString(graph,
    XRT_HEADER_BACKGROUND_COLOR, "orange");
```

Handling Data Types

While most of the data types used in programming XRT/graph and Olectra Chart's 2D Chart control are the same, some differences exist due to differences in the two platforms. XRT/graph and Olectra Chart provides the following cross-platform `typedefs` for these data types that you can use:

Portable typedef	Data Type in XRT/graph	Data Type in Olectra Chart
	`Widget`	`HXRT2D`

Portable typedef	Data Type in XRT/graph	Data Type in Olectra Chart
	`float`	`double`
XrtBoolean	`Boolean`	`BOOL`
XrtColor	`char *`	`COLORREF`
XrtDimension	`Dimension`	`int`
XrtFont	`XmFontList`	`HFONT`
XrtPosition	`Position`	`int`

Cross-Platform Example

XRT/graph includes an example program, *common.c* that can be compiled on both Motif and Windows systems, as shown in Figure 91. It illustrates many of the challenges involved in this kind of cross-platform development. *common.c* is located in *$XRTHOME/src/graph/examples*. To compile the program on a Windows environment:

1. Ensure that Olectra Chart is installed on the Windows system and configured for your Windows development environment. You should be able to compile and run the sample code in Olectra Chart's *\CHART\2D\DEMOS\DLL\SDK* directory.
2. Copy *common.c*, *common.rc*, *olch2d.ico*, *resource.h*, and *makefile.w32* to a directory on the Windows system. If using a 16-bit development environment, also copy *makefile.w16* and *common.def* (the directory specified in the makefile is Olectra Chart's *CHART\2D\COMMON* directory).

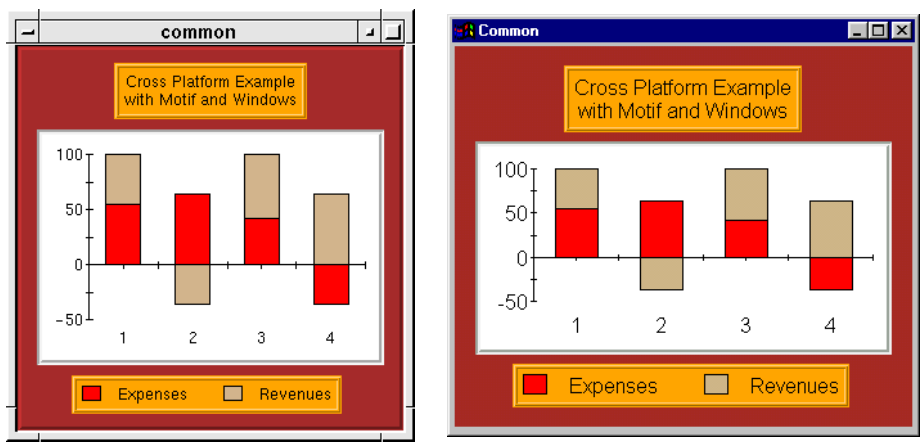


Figure 91 common.c on Motif and Windows platforms

7.8 Customizing Warning Messages

XRT/graph uses *XtAppWarningMsg()* to output warning messages. The Xt error database allows the text of warnings to be customized independent of the application. XRT/graph's messages are provided in *$XRTHOME/include/Xm/XrtGraph.err*.

To modify the warning and error messages, modify a copy of this file and either:

- Add the modified messages to the system message file in */usr/lib/X11/XtErrorDB*; or
- Merge the modified file with the system file at runtime by calling the Xlib routine *XrmMergeDatabases()*.

7.9 Using C++ with XRT/graph

XRT/graph provides a simple C++ interface to the widget. To use it, include the header file *$XRTHOME/include/Xm/XrtGraph.hxx* in your C++ program. This file creates the three classes described below.

Class	Methods
XrtGraph	■ A constructor that calls *XtCreateXrtGraph()*. ■ A destructor that calls *XtDestroyWidget()*. ■ A method for each of the C procedures that take an XRT/graph widget as its first argument. For example, the method for *XrtDrawPS()* is *drawPS()*. ■ A method to set each settable resource. For example, the method to set `XtNxrtType` is *setType()*. ■ A method to retrieve each resource. For example, the method to retrieve `XtNxrtBackgroundColor` is *getBackgroundColor()*. ■ Methods to perform common Xt functions: *addCallback()*, *destroy()*, *getDisplay()*, *getName()*, *getParent()*, *isManaged()*, *isRealized()*, *manageChild()*.
XrtGraphData	■ A constructor that calls one of *XrtDataCreate()*, *XrtDataCreateFromFile()*, or *XrtDataCopy()*. ■ A destructor that calls *XrtDataDestroy()*. ■ A method for most of the C procedures that take a data object handle as its first argument. For example, the method for *XrtDataSetYElement()* is *setYElement()*. ■ Overloaded = operator to perform *XrtDataCopy()*. ■ Casting support for (`XrtDataHandle`).
XrtTextArea	■ A constructor that calls *XrtTextAreaCreate()*. ■ A destructor that calls *XrtTextAreaDestroy()*. ■ A method to set each settable text object attribute. For example, the method to set `XRT_TEXT_ATTACH_TYPE` is *setAttachType()*. ■ A method to retrieve each text object attribute. For example, the method to retrieve `XRT_TEXT_BACKGROUND_COLOR` is *getBackgroundColor()*.

Class	Methods
XrtPointStyle	■ A constructor that calls *XrtPointStyleCreate()*. ■ A destructor that calls *XrtPointStyleDestroy()*. ■ A constructor that retrieves an existing XrtPointStyle by point and set. ■ A method to set each settable pointstyle attribute. For example, the method to retrieve `XRT_POINTSTYLE_DISPLAY` is *setDisplay()*. ■ A method to retrieve each pointstyle attribute. For example, the method to retrieve `XRT_POINTSTYLE_SLICE_OFFSET` is *getSliceOffset()*.

A simple C++ example is found in *$XRTHOME/src/graph/examples/bar2tran_cpp.cxx*.

7.10 Using UIL with XRT/graph

XRT/graph allows you to use OSF's User Interface Language (UIL) to create XRT/graph widgets. Since UIL does not support any datatypes that are not used by standard Motif widgets, some XRT/graph resources cannot be set using UIL, but can be set in an application with an `MrmNcreateCallback`. See Appendix I on page 283 for a complete list of UIL arguments and their datatypes.

To create a graph with UIL:

- Include the XRT/graph UIL header file *XrtGraph.uil*;
- Call *XrtMrmInitialize()* immediately after calling *MrmInitialize()*.

$XRTHOME/src/graph/examples contains a short UIL example program—*simple_uil.c* and *simple_uil.uil*.

7.11 Converting Resource Values to Strings

Resource converters convert resource values from one format to another. This is commonly used to convert values stored in resource files from strings to the format that the widget understands (for example, Booleans and integers). XRT/graph automatically registers its built-in resource converters to convert between strings and all of the XRT/graph-specific resource types (such as `XtRXrtDataStyle` and `XtRXrtAlign`).

XRT/graph also automatically registers "reverse converters" – resource converters that convert from XRT/graph-specific resource types to strings. This is useful for writing resource values to a resource file, or saving a widget's state. Every reverse converter allocates the space necessary to store the conversion result. This string must be freed by the calling procedure after use.

To enable the converters before the creation of a graph widget, call *XtInitializeWidgetClass()*. To enable only the reverse converters, call *XrtSetReverseConverters()*.

The following code fragment shows how these converters could be used to print the value of the legend orientation:

```
void
print_legend_orientation()
{
    XrtAlign    align;
    XrmValue    from, to;

    XrtSetReverseConverters();

    XtVaGetValues(graph,
            XtNxrtLegendOrientation, &align, NULL);
    from.addr = (caddr_t) &align;
    from.size = sizeof(align);
    to.addr = NULL;
    if (XtConvertAndStore(graph, XtRXrtAlign, &from,
                    XtRString, &to))   {
        printf("Legend Orientation: %s\n", to.addr);
        XtFree(to.addr);
    }
}
```

An alternative, and easier way to do this would be via *XrtGetPropString()*:

```
void
print_legend_orientation()
{
    string      string;
    if (XrtGetPropString(graph, XRT_LEGEND_ORIENTATION, &string)) {
        printf("Legend Orientation: %s\n", string);
        XrtFreePropString(string);
    }
}
```

8

PDS Utility Library

XrtPDSImage ■ Registering Image Types
Using Dynamic or Shared Libraries ■ XrtPDSList Object
Procedures and Methods Reference

8.1 XrtPDSImage

An XrtPDSImage is an opaque generic object which contains an image. It also encapsulates some of the image's properties such as source filename, width, 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.

8.1.1 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/graph, 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 8.2.2).

8.2 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
XrtEnableImageType(
    XtPointer   class,
    XtPointer   functions
)
```

where *class* is the widget class (i.e. `xtXrtGraphWidgetClass`) and *functions* is a pointer to an `XrtPDSImageFuncs` structure. This structure is defined as follows:

```
typedef struct {
    XrtPDSImageNameCompareProc  check_name;
    XrtPDSImageHeaderCompareProccheck_header;
    XrtPDSImageCreateProc       create_image;
    XrtPDSImageType             type;
    XrtPDSImageOutputProc       output;
} XrtPDSImageFuncs
```

For example, to register JPEG input and output with the XRT/graph widget:

```
#include <Xm/XrtGraph.h>
#include <Xm/XrtPDSJpeg.h>
XrtEnableImageType((XtPointer)xtXrtGraphWidgetClass,
                   (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 a `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.

8.2.1 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/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` for XPM images or `XrtPDSJpegImageUtils` 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.

8.2.2 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:

```
XrtEnableImageType(xtXrtGraphWidgetClass, TargaImageUtils)
```

Targa images for all the image resources (such as `XtNxrtDataAreaImage`) can now be read. You can also write Targa images by calling *XrtSaveImageGeneric()*:

```
XrtSaveImageGeneric(graph, TARGA_IMAGE_TYPE, "output.tga", NULL)
```

8.2.3 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.

8.3 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. Specifically:

SGI IRIX 5.x

SGI IRIX 6.x

SGI IRIX 6.x 64-bit

HP 9000 Series 700/800

SPARC SunOS 4.x

Alpha/OSF

SPARC Solaris 2.x

SPARC Solaris 7.x 64-bit

SPARC Solaris 7.x

IBM RS6000 AIX 4.x

IBM RS6000 AIX 4.x 64-bit

8.4 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.

8.4.1 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.

8.4.2 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.

8.4.3 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 8.4.6 on page 155.)

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.

8.4.4 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 8.4.6 on page 155 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);
```

8.4.5 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()*.

8.4.6 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 8.4.4 on page 151.

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.

8.4.7 User-Defined Data

The XrtPDSList object enables you to define user data to be associated with the list. To define user data, pass a pointer to the user data to the *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.

8.5 Procedures and Methods Reference

8.5.1 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 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.

8.5.2 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 alist.

```
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 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
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

Reference Appendices

A

Resource Reference

Widget Synopsis
XRT/graph Resources
Inherited Resources

This appendix lists all of the XRT/graph resources in alphabetical order. 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()*).

A.1 Widget Synopsis

Include File:	*<Xm/XrtGraph.h>*
Class Name:	XtXrtGraph
Class Hierarchy:	Core → XmPrimitive → XtXrtGraph
Class Pointer:	xtXrtGraphWidgetClass
Instantiation:	`widget = XtCreateWidget(name, xtXrtGraphWidgetClass...)`
Functions/Macros:	*XtCreateXrtGraph()*, *XtIsXrtGraph()*

A.2 XRT/graph Resources

XtNxrtAngleUnit **XrtAngleUnit** **XRT_ANGLE_DEGREES** **CSG**

Specifies the units used to interpret data X-values and X-axis resources. Used primarily in polar graphs, also radar and filled radar charts. Valid values are `XRT_ANGLE_DEGREES`, `XRT_ANGLE_RADIANS`, and `XRT_ANGLE_GRADS`.

XtNxrtAxisBoundingBox **Boolean** **False** **CSG**

When `True`, XRT/graph will draw a box inside the graph area bounding the graph's data area, that is, the axes, all displayed data, and the gridlines. Not used in polar or pie charts.

XtNxrtAxisFont **XmFontList** **"fixed"** **CSG**

Specifies the font to be used for the axes numbering, annotation and titles. It is also used for text areas when they are created without specifying a font.

XtNxrtBackgroundColor **char *** **"White"** **CSG**

Specifies the window background color. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification. This color will be inherited as the default background color for the graph, data area, header, footer and legend.

Setting this color to anything other than black will result in white when running on a monochrome Server.

XtNxrtBarClusterOverlap **int** **100** **CSG**

Specifies the overlap between bars in a cluster, as a percentage of the bar width. A setting of 0 will cause the bars in the cluster to completely overlap each other. A setting of 100 will cause the bars to be fitted next to each other. The value must be between 0 and 200. The actual bar width is determined at run time by XRT/graph and depends on a number of factors, including the current window size. Since stacking bars have only one bar in each cluster, this resource is meaningless when `XtNxrtType` is `XRT_TYPE_STACKING_BAR`.

XtNxrtBarClusterWidth **int** **50** **CSG**

Specifies the amount of space to be used for the bars in a cluster, as a percentage of the total amount of space assigned to the cluster. The value must be between 0 and 100. Setting the value to 100 will cause the bars to abut the bars in the next cluster.

XtNxrtBorder **XrtBorder (enum)** **XRT_BORDER_3D_OUT** **CSG**

Specifies the style of border used to enclose the graph. Valid styles are `XRT_BORDER_NONE`, `XRT_BORDER_PLAIN`, `XRT_BORDER_3D_IN`, `XRT_BORDER_3D_OUT`, `XRT_BORDER_ETCHED_IN`, `XRT_BORDER_ETCHED_OUT`, `XRT_BORDER_FRAME_IN`, `XRT_BORDER_FRAME_OUT`, `XRT_BORDER_BEVEL`, and `XRT_BORDER_SHADOW`. Use the `XmNshadowThickness` resource to specify the border width.

XtNxrtBubbleMethod **XrtBubbleMethod (enum)** **XRT_BUBBLE_METHOD_DIAMETER**

Specifies the method used to encode the additional information presented in Bubble charts. Possible values are `XRT_BUBBLE_METHOD_DIAMETER` and `XRT_BUBBLE_METHOD_AREA`. The first value causes the information to be encoded by varying the diameter of the point symbol. The second value causes the information to be encoded by varying the area of the point symbol.

XtNxrtBubbleMax **float** **20.0**
XtNxrtBubbleMin **float** **5.0** **CSG**

Specifies the minimum and the maximum diameter/area of the point symbols as a percentage of the diameter/area of the plot area. When the value of `XtNxrtBubbleMin` is larger than the value of `XtNxrtBubbleMax`, the point symbol diameter/area decreases as bubble values

increase. Since the diameter/area of point symbols is dependent on the *range* of bubble values, bubble values do not need to be strictly positive or negative.

Note: Remember to use *XrtFloatToArgVal()* when setting resources of type float.

XtNxrtCandleComplex **Boolean** **False** **CSG**

Only referenced if the graph is of type `XRT_TYPE_CANDLE`. If `False`, XRT/graph will use the data style from the first set of each logical set to determine the appearance of the candle. If `True`, all four set's data styles will be used to determine the appearance.

XtNxrtData **XrtDataHandle** **none** **CSG**
XtNxrtData2 **XrtDataHandle** **none** **CSG**

Specifies the data to be displayed. The data object specified should be allocated using the procedure *XrtDataCreate()* or *XrtDataCreateFromFile()*. `XtNxrtData2` is used to specify the data for the second graph in a combination graph. The Data resources, unlike all other XRT/graph resources, are *not* copied by the widget. It is the application's responsibility to allocate and deallocate these data objects.

XtNxrtDataAreaBackgroundColor **char *** **NULL** **CSG**

Specifies the background color of the graph's data area. When `XtNxrtDataAreaBackgroundColor` and `XtNxrtDataAreaImage` are both `NULL`, the data area background is transparent. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

This resource is ignored in pie charts. Setting this resource to anything other than black results in white when running on a monochrome server.

XtNxrtDataAreaForegroundColor **char *** **NULL** **CSG**

Specifies the color of the boundary drawn around bars, filled areas, and pie slices (displayed when `XtNxrtGraphShowOutlines` is `True`). When `NULL`, the graph foreground color is used. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than white will result in black when running on a monochrome server.

XtNxrtDataAreaImage **XrtImage** **NULL** **CSG**

Specifies the image to display in the background of the graph's data area. See Background Images on page 91 for more information on setting background images.

XtNxrtDataAreaImageLayout **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED** **CSG**

Specifies the positioning and scaling of the background image. Valid values are: `XRT_IMAGELAYOUT_CENTERED` and `XRT_IMAGELAYOUT_TILED`. See Background Images on page 91 for more information on setting background images.

XtNxrtDataStyles **XrtDataStyle **** **dynamic** **CSG**
XtNxrtDataStyles2 **XrtDataStyle **** **dynamic** **CSG**

The `XrtDataStyle` structure defines a line pattern, fill pattern, color, line width, point style, point size and point color for each set of data. XRT/graph will only create default `XrtData` styles as required at drawing time throughout the life of the graph object.

Use `XtNxrtDataStyles` to determine what data styles are currently being used, or to define the data styles to be used. The list of styles is terminated with a `NULL` pointer.

When used with *XtGetValues()*, a pointer to an array of `XrtDataStyle` structures is returned. The length of the array is the larger of: the current number of sets in the corresponding data object, and the highest index of any data style specified so far by the program.

`XtNxrtDataStyles2` is used to access the second graph's data styles in a combination graph.

XtNxrtDataStylesUseDefault **Boolean** **True** **CSG**
XtNxrtDataStyles2UseDefault **Boolean** **True** **CSG**

When `True`, data styles values are determined by XRT/graph at run time. This is useful when the type of Server (color or monochrome) is not known at compile time and the programmer does not want to deal with setting different data styles depending on Server type. When explicit data styles are provided, XRT/graph sets this Boolean to `False`.

`XtNxrtDataStyles2UseDefault` applies to the second graph's data styles in a combination graph.

XtNxrtDebug **Boolean** **False** **CSG**

Specifies whether to send warning messages to *stderr*. XRT/graph uses the Xt Error Database facility to allow customization of message text. If set to `False`, only resource conversion errors are output.

XtNxrtDoubleBuffer **Boolean** **True** **CSG**

When `True`, graph updates are first rendered into an off-screen area, then copied to the display area. This reduces flashing as similar graphs are displayed on the screen. Setting to `False` will cause graphs to be rendered directly to the display and will result in less memory utilization by the X Server.

XtNxrtExtraDefaultDataStyles **Boolean** **False** **CSG**

Specifies whether the additional 8 default data styles are available or not. If there's a concern about version incompatibility, this should not be enabled. Changing this resource after create has no effect.

XtNxrtExposeCallback **XtCallbackList** **NULL** **CSG**

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

callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                  graph,
    caddr_t                 client_data,
    XrtCallbackStruct       *call_data
    )
```

where XrtCallbackStruct is defined as:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    Window      window;     /* Read-only */
    Region      region;     /* Read-only */
    int         change;     /* Read-only */
} XrtCallbackStruct;
```

reason is always XmCR_EXPOSE; *region* is the area of the widget that required repainting (NULL if no area was repainted); *change* describes which areas of the widget were actually repainted, constructed from the following values:

```
#define XRT_GRAPH_CHANGED       0x01
#define XRT_HEADER_CHANGED      0x02
#define XRT_FOOTER_CHANGED      0x04
#define XRT_LEGEND_CHANGED      0x08
#define XRT_DATA_CHANGED        0x09 /* graph and legend */
#define XRT_MARKER_CHANGED      0x10
#define XRT_ALL_CHANGED         0xFF
```

Note that a change to data requires the graph and legend to be redrawn. Also note that if a call to *XtSetValues()* on the widget is made from within an expose callback, the list of expose callbacks is not called, to avoid an infinite loop.

XtNxrtFooterAdjust **XrtAdjust (enum)** **XRT_ADJUST_CENTER** **CSG**

Specifies how multiple footer text lines should be adjusted within the footer area. Must be one of XRT_ADJUST_CENTER, XRT_ADJUST_LEFT or XRT_ADJUST_RIGHT.

XtNxrtFooterBackgroundColor **char *** **NULL** **CSG**

Specifies the footer area background color. When XtNxrtFooterBackgroundColor and XtNxrtFooterImage are both NULL, the footer background is transparent. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than black will result in white when running on a monochrome Server.

XtNxrtFooterBorder **XrtBorder (enum)** **XRT_BORDER_NONE** **CSG**

Specifies the style of border used to identify the footer area. Valid styles are XRT_BORDER_NONE, XRT_BORDER_PLAIN, XRT_BORDER_3D_IN, XRT_BORDER_3D_OUT, XRT_BORDER_ETCHED_IN, XRT_BORDER_ETCHED_OUT, XRT_BORDER_FRAME_IN, XRT_BORDER_FRAME_OUT, XRT_BORDER_BEVEL, and XRT_BORDER_SHADOW.

XtNxrtFooterBorderWidth **Dimension** **2** **CSG**

Specifies the width of the footer area border in pixels. Must be between 0 and 20.

XtNxrtFooterFont **XmFontList** **"fixed"** **CSG**

Specifies the font to use for the footer strings.

XtNxrtFooterForegroundColor **char *** **NULL** **CSG**

Specifies the footer area foreground color. When `NULL`, the window foreground color is used. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than white will result in black when running on a monochrome Server.

XtNxrtFooterImage **XrtImage** **NULL** **CSG**

Specifies the filename of an image to display in the background of the footer area. See Background Images on page 91 for more information on setting background images.

XtNxrtFooterImageLayout **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED** **CSG**

Specifies the positioning and layout of the background image. Valid values are: `XRT_IMAGELAYOUT_CENTERED` and `XRT_IMAGELAYOUT_TILED`. See Background Images on page 91 for more information on setting background images.

XtNxrtFooterImageMinimumSize **Boolean** **False** **CSG**

Specifies the minimum size for a label that corresponds to the size of the image. This is not available for non-text related images (for example, the graph area background image). See Background Images on page 91 for more information on setting background images.

XtNxrtFooterStrings **char **** **none** **CSG**

Specifies the text to be displayed in the footer area.

XtNxrtFooterHeight **Dimension** **dynamic** **G**
XtNxrtFooterWidth **Dimension** **dynamic** **G**

Contains the width and height of the footer area in pixels.

XtNxrtFooterX **Position** **centered under graph** **CSG**
XtNxrtFooterY **Position** **centered under graph** **CSG**

Specifies the (X,Y) coordinates of the top left corner of the footer area. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrtFooterXUseDefault **Boolean** **True** **CSG**
XtNxrtFooterYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtFooterX` and `XtNxrtFooterY` values are determined by XRT/graph when the graph is rendered. When explicit `XtNxrtFooterX` and `XtNxrtFooterY` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrtFooterX` or `XtNxrtFooterY` values.

XtNxrtForegroundColor **char *** **"Black"** **CSG**

Specifies the window foreground color which is used for header, footer and legend text, and for drawing the axes. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification. This color will be inherited as the default foreground color for the header, footer, and legend text, as well as the graph axes.

Setting this color to anything other than white will result in black when running on a monochrome Server.

XtNxrtFrontDataset **int** **XRT_DATASET1** **CSG**

Specifies which dataset is displayed in front (i.e. closer to the viewer) in a combination graph. Valid values are `XRT_DATASET1` and `XRT_DATASET2`.

XtNxrtGraph3dShading **XrtShading** **XRT_SHADING_COLOR** **CSG**

Specifies how to color the sides of graphs with a 3D effect.[1] Valid values are: `XRT_SHADING_COLOR`, which creates two additional colors for each bar or pie slice; `XRT_SHADING_DITHERED`, which uses dithering to mimic light/dark shading without allocating additional colors; or `XRT_SHADING_NONE`, which turns off shading.

XtNxrtGraphBackgroundColor **char *** **NULL** **CSG**

Specifies the graph area background color. When `XtNxrtGraphBackgroundColor` and `XtNxrtGraphImage` are both `NULL`, the graph area is transparent. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than black will result in white when running on a monochrome Server.

XtNxrtGraphBorder **XrtBorder (enum)** **XRT_BORDER_NONE** **CSG**

Specifies the style of border used to identify the graph area. Valid styles are `XRT_BORDER_NONE`, `XRT_BORDER_PLAIN`, `XRT_BORDER_3D_IN`, `XRT_BORDER_3D_OUT`, `XRT_BORDER_ETCHED_IN`, `XRT_BORDER_ETCHED_OUT`, `XRT_BORDER_FRAME_IN`, `XRT_BORDER_FRAME_OUT`, `XRT_BORDER_BEVEL`, and `XRT_BORDER_SHADOW`.

XtNxrtGraphBorderWidth **Dimension** **2** **CSG**

Specifies the width of the graph area border in pixels. Must be between 0 and 20.

XtNxrtGraphDepth **int** **0** **CSG**

Specifies the apparent graph depth as a percentage of graph width when a 3D effect is desired. Graph depth, and either inclination or rotation must be non-zero for a 3D effect to appear. Must be between 0 and 500. Only bar, stacking bar, and pie charts will display a 3D effect.

1. 3D effect is controlled by `XtNxrtGraphDepth`, `XtNxrtGraphInclination`, and `XtNxrtGraphRotation`.

XtNxrtGraphForegroundColor **char *** **NULL** **CSG**

Specifies the graph area foreground color. When NULL, the window foreground color is used. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than white will result in black when running on a monochrome Server.

XtNxrtGraphImage **XrtImage** **NULL** **CSG**

Specifies the filename of an image to display in the background of the graph area. See Background Images on page 91 for more information on setting background images.

XtNxrtGraphImageLayout **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED** **CSG**

Specifies the positioning and scaling of the background image. Valid values are: XRT_IMAGELAYOUT_CENTERED and XRT_IMAGELAYOUT_TILED. See Background Images on page 91 for more information on setting background images.

XtNxrtGraphInclination **int** **0** **CSG**

Specifies the apparent degree of inclination of the eye-point above the X-axis when a 3D effect is desired. Graph depth, and either inclination or rotation must be non-zero for a 3D effect to appear. Must be between -45 and 45. Only bar, stacking bar, and pie charts will display a 3D effect.

XtNxrtGraphMarginBottom **Position** **dynamic** **CSG**

Specifies the pixel offset (with respect to the bottom of the graph area) that the data area should be drawn. If XtNxrtGraphMarginBottomUseDefault is True, XRT/graph calculates the value of this resource based on the space required for the annotation and titles. A side-effect of setting this resource is that the corresponding UseDefault resource is set to False. This resource must be between XRT_MIN_MARGIN and XRT_MAX_MARGIN.

XtNxrtGraphMarginBottomUseDefault **Boolean** **True** **CSG**

When True, XtNxrtGraphMarginBottom is determined by XRT/graph at render-time. When you set XtNxrtGraphMarginBottom, this resource is set to False. You cannot set this resource to False unless a value for XtNxrtGraphMarginBottom has been provided or calculated.

XtNxrtGraphMarginLeft **Position** **dynamic** **CSG**

Specifies the pixel offset (with respect to the left side of the graph area) that the data area should be drawn. If XtNxrtGraphMarginLeftUseDefault is True, XRT/graph calculates the value of this resource based on the space required for the annotation and titles. A side-effect of setting this resource is that the corresponding UseDefault is set to False.

XtNxrtGraphMarginLeftUseDefault **Boolean** **True** **CSG**

When True, XtNxrtGraphMarginLeft is determined by XRT/graph at render-time. When you set XtNxrtGraphMarginLeft, this resource is set to False. You cannot set this resource to False unless a value for XtNxrtGraphMarginLeft has been provided or calculated.

XtNxrtGraphMarginRight **Position** **dynamic** **CSG**

Specifies the pixel offset (with respect to the right side of the graph area) that the data area should be drawn. If `XtNxrtGraphMarginRightUseDefault` is `True`, XRT/graph calculates the value of this resource based on the space required for the annotation and titles. A side-effect of setting this resource is that the corresponding `UseDefault` is set to `False`.

XtNxrtGraphMarginRightUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtGraphMarginRight` is determined by XRT/graph at render-time. When you set `XtNxrtGraphMarginRight`, this resource is set to `False`. You cannot set this resource to `False` unless a value for `XtNxrtGraphMarginRight` has been provided or calculated.

XtNxrtGraphMarginTop **Position** **dynamic** **CSG**

Specifies the pixel offset (with respect to the top of the graph area) that the data area should be drawn. If `XtNxrtGraphMarginTopUseDefault` is `True`, XRT/graph calculates the value of this resource based on the space required for the annotation and titles. A side-effect of setting this resource is that the corresponding `UseDefault` is set to `False`.

XtNxrtGraphMarginTopUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtGraphMarginTop` is determined by XRT/graph at render-time. When you set `XtNxrtGraphMarginTop`, this resource is set to `False`. You cannot set this resource to `False` unless a value for `XtNxrtGraphMarginTop` has been provided or calculated.

XtNxrtGraphRotation **int** **0** **CSG**

Specifies the apparent degree of rotation of the eye-point to the right of the Y-axis when a 3D effect is desired. Graph depth, and either inclination or rotation must be non-zero for a 3D effect to appear. Must be between -45 and 45. Only bar and stacking bar charts will display a 3D effect.

XtNxrtGraphShowOutlines **Boolean** **True** **CSG**

Specifies whether to draw the boundary line around bars, filled areas, and pie slices. When set to `True`, XRT/graph draws the boundary line (in the color specified by `XtNxrtDataAreaForegroundColor`). When set to `False`, no boundary line is drawn.

XtNxrtGraphHeight **Dimension** **dynamic** **CSG**
XtNxrtGraphWidth **Dimension** **dynamic** **CSG**

Specifies the width and height of the graph area in pixels. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

The graph area is the area surrounding the axis and annotation, but does not include the header, footer and legend. To adjust the size of the whole graph widget use the `XtNwidth` and `XtNheight` resources.

XtNxrtGraphWidthUseDefault **Boolean** **True** **CSG**
XtNxrtGraphHeightUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtGraphWidth` and `XtNxrtGraphHeight` values are determined by XRT/graph at render-time. When explicit `XtNxrtGraphWidth` and `XtNxrtGraphHeight` values

are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrtGraphWidth` or `XtNxrtGraphHeight` values.

XtNxrtGraphX **Position** **dynamic** **CSG**
XtNxrtGraphY **Position** **dynamic** **CSG**

Specifies the (X,Y) coordinates of the top left corner of the graph area. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrtGraphXUseDefault **Boolean** **True** **CSG**
XtNxrtGraphYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtGraphX` and `XtNxrtGraphY` values are determined by XRT/graph at render-time. When explicit `XtNxrtGraphX` and `XtNxrtGraphY` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrtGraphX` or `XtNxrtGraphY` values.

XtNxrtHeaderAdjust **XrtAdjust (enum)** **XRT_ADJUST_CENTER** **CSG**

Specifies how multiple header text lines should be adjusted within the header area. Must be one of `XRT_ADJUST_CENTER`, `XRT_ADJUST_LEFT` or `XRT_ADJUST_RIGHT`.

XtNxrtHeaderBackgroundColor **char *** **NULL** **CSG**

Specifies the header area background color. When `XtNxrtHeaderBackgroundColor` and `XtNxrtHeaderImage` are both `NULL`, the header is transparent. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than black will result in white when running on a monochrome Server.

XtNxrtHeaderBorder **XrtBorder (enum)** **XRT_BORDER_NONE** **CSG**

Specifies the style of border used to identify the header area. Valid styles are `XRT_BORDER_NONE`, `XRT_BORDER_3D_IN`, `XRT_BORDER_3D_OUT`, `XRT_BORDER_ETCHED_IN`, `XRT_BORDER_ETCHED_OUT`, `XRT_BORDER_SHADOW`, and `XRT_BORDER_PLAIN`.

XtNxrtHeaderBorderWidth **Dimension** **2** **CSG**

Specifies the width of the header area border in pixels. Must be between 0 and 20.

XtNxrtHeaderFont **XmFontList** **"fixed"** **CSG**

Specifies the font to use for the header strings.

XtNxrtHeaderForegroundColor **char *** **NULL** **CSG**

Specifies the header area foreground color. When `NULL`, the window foreground color is used. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than white will result in black when running on a monochrome Server.

XtNxrtHeaderImage **XrtImage** **NULL** **CSG**

Specifies the filename of an image to display in the background of the header area. See Background Images on page 91 for more information on setting background images.

XtNxrtHeaderImageLayout **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED** **CSG**

Specifies the positioning and scaling of the background image. Valid values are: `XRT_IMAGELAYOUT_CENTERED` and `XRT_IMAGELAYOUT_TILED`. See Background Images on page 91 for more information on setting background images.

XtNxrtHeaderImageMinimumSize **Boolean** **False** **CSG**

Specifies the minimum size for a label that corresponds to the size of the image. This is not available for non-text related images (for example, the graph area background image). See Background Images on page 91 for more information on setting background images.

XtNxrtHeaderStrings **char **** **none** **CSG**

Specifies the text to be displayed in the header area.

XtNxrtHeaderHeight **Dimension** **dynamic** **G**
XtNxrtHeaderWidth **Dimension** **dynamic** **G**

Contains the height and width of the header area in pixels.

XtNxrtHeaderX **Position** **centered above graph** **CSG**
XtNxrtHeaderY **Position** **centered above graph** **CSG**

Specifies the (X,Y) coordinates of the top left corner of the header area. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrtHeaderXUseDefault **Boolean** **True** **CSG**
XtNxrtHeaderYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtHeaderX` and `XtNxrtHeaderY` values are determined by XRT/graph at render-time. When explicit `XtNxrtHeaderX` and `XtNxrtHeaderY` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrtHeaderX` or `XtNxrtHeaderY` values.

XtNxrtHiLoCloseShow **Boolean** **True** **CSG**

Specifies whether to display the close tick mark when the graph type is `XRT_TYPE_HILO_OPEN_CLOSE`.

XtNxrtHiLoOpenCloseFullWidth **Boolean** **False** **CSG**

Specifies whether to draw open/close tick marks across the full width, useful for creating error bar charts. Only has effect when the graph type is `XRT_TYPE_HILO_OPEN_CLOSE`.

XtNxrtHiLoOpenShow **Boolean** **True** **CSG**

Specifies whether to display the open tick mark when the graph type is `XRT_TYPE_HILO_OPEN_CLOSE`.

XtNxrtImage **XrtImage** **NULL** **CSG**

Specifies the filename of an image to display in the background of the widget. See Background Images on page 91 for more information on setting background images.

XtNxrtImageLayout **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED** **CSG**

Specifies the positioning and scaling of the background image. Valid values are: `XRT_IMAGELAYOUT_CENTERED` and `XRT_IMAGELAYOUT_TILED`. See Background Images on page 91 for more information on setting background images.

XtNxrtInvertOrientation **Boolean** **False** **CSG**

Inverts the normal interpretation of X and Y. When `True`, causes X-values to be graphed against the vertical axis, and Y-values to be graphed against the horizontal axis. Bar and stacking bar graphs will render horizontally. This resource has no effect on pie, polar, radar, or filled radar charts.

XtNxrtIsStacked **Boolean** **False** **CSG**
XtNxrtIsStacked2 **Boolean** **False** **CSG**

Specifies whether or not to use stacking for applicable graph types. Applicable graph types are plot, area, polar, radar and filled radar. Bar charts are not affected, since there is already a stacking bar graph type. **NOTE:** Stacking requires the array data type.

XtNxrtLegendAnchor **XrtAnchor (enum)** **XRT_ANCHOR_EAST** **CSG**

Specifies where to anchor the legend to the window. Valid values are: `XRT_ANCHOR_NORTH`, `XRT_ANCHOR_SOUTH`, `XRT_ANCHOR_EAST`, `XRT_ANCHOR_WEST`, `XRT_ANCHOR_NORTHWEST`, `XRT_ANCHOR_NORTHEAST`, `XRT_ANCHOR_SOUTHWEST`, and `XRT_ANCHOR_SOUTHEAST`.

XtNxrtLegendBackgroundColor **char *** **NULL** **CSG**

Specifies the legend area background color. When `XtNxrtLegendBackgroundColor` and `XtNxrtLegendImage` are both `NULL`, the legend background is transparent. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than black will result in white when running on a monochrome Server.

XtNxrtLegendBorder **XrtBorder (enum)** **XRT_BORDER_NONE** **CSG**

Specifies the style of border used to identify the legend area. Valid styles are `XRT_BORDER_NONE`, `XRT_BORDER_PLAIN`, `XRT_BORDER_3D_IN`, `XRT_BORDER_3D_OUT`, `XRT_BORDER_ETCHED_IN`, `XRT_BORDER_ETCHED_OUT`, `XRT_BORDER_FRAME_IN`, `XRT_BORDER_FRAME_OUT`, `XRT_BORDER_BEVEL`, and `XRT_BORDER_SHADOW`.

XtNxrtLegendBorderWidth **Dimension** **2** **CSG**

Specifies the width of the legend area border in pixels. Must be between 0 and 20.

XtNxrtLegendFont **XmFontList** **"fixed"** **CSG**

Specifies the font to be used in the legend.

XtNxrtLegendForegroundColor **char *** **NULL** **CSG**

Specifies the legend area foreground color. When `NULL`, the window foreground color is used. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than white will result in black when running on a monochrome Server.

XtNxrtLegendImage **XrtImage** **NULL** **CSG**

Specifies the filename of an image to display in the background of the legend area. See Background Images on page 91 for more information on setting background images.

XtNxrtLegendImageLayout **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED** **CSG**

Specifies the positioning and scaling of the background image. Valid values are: `XRT_IMAGELAYOUT_CENTERED` and `XRT_IMAGELAYOUT_TILED`. See Background Images on page 91 for more information on setting background images.

XtNxrtLegendOrientation **XrtAlign (enum)** **XRT_ALIGN_VERTICAL** **CSG**

Specifies the orientation of the legend. Valid values are `XRT_ALIGN_HORIZONTAL` and `XRT_ALIGN_VERTICAL`.

XtNxrtLegendReversed **Boolean** **False** **CSG**

Allows the user to specify that the set entries in the legend should be displayed in reverse order.

XtNxrtLegendShow **Boolean** **True** **CSG**

Determines if the legend will be displayed.

XtNxrtLegendTitle **char *** **NULL** **CSG**

Allows the user to specify the legend title. The title is limited to a single line, is centered above the legend and uses the legend font.

XtNxrtLegendHeight **Dimension** **dynamic** **G**
XtNxrtLegendWidth **Dimension** **dynamic** **G**

Contains the height and width of the legend area in pixels.

XtNxrtLegendX **Position** **depends on anchor** **CSG**
XtNxrtLegendY **Position** **depends on anchor** **CSG**

Specifies the (X,Y) coordinates of the top left corner of the legend area. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrtLegendXUseDefault **Boolean** **True** **CSG**
XtNxrtLegendYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrtLegendX` and `XtNxrtLegendY` values are determined by XRT/graph at render-time. When explicit `XtNxrtLegendX` and `XtNxrtLegendY` values are provided,

XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrtLegendX` or `XtNxrtLegendY` values.

XtNxrtMapCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application of the graph coordinates of a user event. Map callbacks are triggered by the *Map()* action[1], which uses the *XrtMap()* procedure. A pointer to an `XrtMapCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                  graph,
    caddr_t                 client_data,
    XrtMapCallbackStruct    *call_data
    )
```

where `XrtMapCallbackStruct` is defined as:

```
typedef struct {
    int             reason;     /* Read-only */
    XEvent          *event;     /* Read-only */
    String          *params;    /* Read-only */
    Cardinal        nparams;    /* Read-only */
    XrtRegion       region;     /* Read-only */
    XrtMapResult    map;        /* Read-only */
} XrtMapCallbackStruct;
```

reason is always `XRT_REASON_MAP`; *region* is the area of the widget in which the event occurred; *map* contains the graph coordinates calculated by *XrtMap()*.

XtNxrtMarkerDataset **int** **XRT_DATASET1** **CSG**

Specifies which dataset the markers should be positioned through in combination graphs. Change `XtNxrtMarkerDataset` to `XRT_DATASET2` to position through the second dataset using `XtNxrtXMarkerPoint`, `XtNxrtXMarkerSet` and `XtNxrt[XY]Marker`.

XtNxrtMarkerDataStyle **XrtDataStyle *** **black, dashed** **CSG**

Specifies the `XrtDataStyle` for both the X- and Y-markers. In particular, line style, color and width elements of the `XrtDataStyle` structure are used. By default, XRT/graph uses a black, 1-pixel wide dashed line for the markers.

This resource is used for a marker only when the `XtNxrt[XY]MarkerDataStyle` resource has not been explicitly set.

XtNxrtMarkerDataStyleUseDefault **Boolean** **True** **CSG**

When `True`, the default marker data style will be used for both markers. When an explicit data style is provided, XRT/graph sets this resource to `False`.

XtNxrtModifyCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when starting and finishing user interaction (scale, zoom, translate, or rotate). Modify callbacks are triggered by the

1. No default translation is bound to this action.

ModifyStart() and *ModifyEnd()* actions. A pointer to an XrtModifyCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                  graph,
    caddr_t                 client_data,
    XrtModifyCallbackStruct *call_data
    )
```

where XrtModifyCallbackStruct is defined as:

```
typedef struct {
    int          reason;    /* Read-only */
    XEvent       *event;    /* Read-only */
    String       *params;   /* Read-only */
    Cardinal     nparams;   /* Read-only */
    Boolean      doit;
} XrtModifyCallbackStruct;
```

reason is either XRT_REASON_MODIFY_START or XRT_REASON_MODIFY_END; *doit* is a flag that determines (when *reason* is XRT_REASON_MODIFY_START) whether to allow the interaction to take place. To disable all user interaction, set *doit* to False in your callback procedure when *reason* is XRT_REASON_MODIFY_START.

XtNxrtOtherDataStyle **XrtDataStyle *** **depends on Server** **CSG**

Specifies the XrtDataStyle to be used when rendering the *other* slice on pie charts. In particular, color and fill pattern elements of the XrtDataStyle structure are used. By default, XRT/graph will generate a data style that is different from those used to represent data (assuming XtNxrtDataStylesUseDefault is True.)

XtNxrtOtherDataStyleUseDefault **Boolean** **True** **CSG**

When True, the *other* data style is determined by XRT/graph at run time. It is guaranteed to be unique if XtNxrtDataStylesUseDefault is True. This is useful when the type of Server (color or monochrome) is not known at compile time and the programmer does not want to deal with setting different data styles depending on Server type. When an explicit other data style is provided, XRT/graph sets this Boolean to False.

XtNxrtOtherLabel **char *** **"Other"** **CSG**

Specifies the label of the *other* slice in a pie chart. By default, the *other* slice is labeled "Other". Attempting to set this resource to an invalid value (such as NULL, or a zero-length string) results in XRT/graph using the string "Other".

XtNxrtPickCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application of the graph data values closest to the pixel coordinates of a user event. Pick callbacks are triggered by the *Pick()* action[1], which uses the *XrtPick()* procedure. A pointer to an XrtPickCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

1. No default translation is bound to this action.

```
void
CallbackProcedure(
    Widget                  graph,
    caddr_t                 client_data,
    XrtPickCallbackStruct   *call_data
    )
```

where XrtPickCallbackStruct is defined as:

```
typedef struct {
    int             reason; /* Read-only */
    XEvent          *event; /* Read-only */
    String          *params;/* Read-only */
    Cardinal        nparams;/* Read-only */
    XrtRegion       region; /* Read-only */
    XrtPickResult   pick;   /* Read-only */
} XrtPickCallbackStruct;
```

reason is always XRT_REASON_PICK; *region* is the area of the widget in which the event occurred; *pick* contains the graph data values calculated by *XrtPick()*.

XtNxrtPieMinSlices **int** **1** **CSG**

Specifies the minimum number of slices (including the *other* slice) to attempt to display in pie charts.

XtNxrtPieOrder **XrtPieOrder (enum)** **XRT_PIEORDER_DATA_ORDER** **CSG**

Specifies the order that pie slices should be displayed. Must be one of XRT_PIEORDER_DESCENDING, XRT_PIEORDER_ASCENDING or XRT_PIEORDER_DATA_ORDER.

XtNxrtPieStartAngle **float** **135.0** **CSG**

Specifies the angle at which pies start drawing. By default, all pies start drawing at 135.0 degrees.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtPieThresholdMethod **XrtPieThresholdMethod** **XRT_PIE_SLICE_CUTOFF** **CSG**

Specifies the threshold method used to determine which point values will be grouped into the *other* slice in pie charts. Must be either XRT_PIE_SLICE_CUTOFF or XRT_PIE_PERCENTILE.

If XRT_PIE_SLICE_CUTOFF, all point values with a percentage less than the value of XtNxrtPieThresholdValue will be grouped into the *other* slice.

If XRT_PIE_PERCENTILE, as many of the smallest points as necessary are grouped together into the *other* slice, providing that the *other* slice is less than or equal to the XtNxrtPieThresholdValue. For example, if you want the *other* slice to be no more than 5% of the total, set XtNxrtPieThresholdMethod to XRT_PIE_PERCENTILE, and set XtNxrtPieThresholdValue to 5.0.

Note that XtNxrtPieMinSlices has precedence over XtNxrtPieThresholdMethod.

XtNxrtPieThresholdValue **float** **0.0** **CSG**

Specifies the threshold value associated with XtNxrtPieThresholdMethod. Must be between 0.0 and 100.0. Set to 0.0 to disable *other* slice creation.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtPointLabels **char **** **NULL** **CSG**
XtNxrtPointLabels2 **char **** **NULL** **CSG**

Defines a list of text strings used to label each point when `XtNxrtXAnnotationMethod` is `XRT_ANNO_POINT_LABELS`. When Point-labels are being used, XRT/graph will:

- Display a label at each point's X-value.
- Ignore the `XtNxrtXAxisLabelFormat`, `XtNxrtXNumMethod`, `XtNxrtXNum`, and `XtNxrtXPrecision` resources.

Setting this resource has no effect unless `XtNxrtXAnnotationMethod` is set to `XRT_ANNO_POINT_LABELS`. To clear all Point-labels, set this resource to `NULL`.

`XtNxrtPointLabels2` is used only if the Y2-axis is the primary axis (i.e. `XtNxrtData` is `NULL`).

XtNxrtPolarAxisAllowNegatives **Boolean** **False** **CSG**

When set to `True`, allows negative values (a negative origin) for polar charts.

XtNxrtPolarHalfRange **Boolean** **False** **CSG**

Specifies the X-axis range used for polar graphs. When set to `False`, the default X-axis annotation ranges from 0 to 360 degrees. When set to `True`, the annotation ranges from -180 to 180 degrees. The actual range uses the units specified by `XtNxrtAngleUnit`.

XtNxrtRepaint **Boolean** **True** **CSG**

When this resource is `True`, any changes made to the widget are rendered immediately. To batch changes to the widget, set this resource to `False`, make the changes, and then set this resource to `True` again.

XtNxrtResizeCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when XRT/graph has resized the graph. A pointer to an `XrtCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                graph,
    caddr_t               client_data,
    XrtCallbackStruct     *call_data
    )
```

where `XrtCallbackStruct` is defined as:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    Window      window;     /* Read-only */
    Region      region;     /* Read-only */
    int         change;     /* Read-only */
} XrtCallbackStruct;
```

reason is always `XmCR_RESIZE`; *event, region* and *change* are always `NULL`.

XtNxrtRotateCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when XRT/graph is about to rotate a 3D bar or pie chart. Rotate callbacks are triggered by the *RotateUpdate()* action. A

pointer to an XrtRotateCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                  graph,
    caddr_t                 client_data,
    XrtRotateCallbackStruct *call_data
    )
```

where XrtRotateCallbackStruct is defined as:

```
typedef struct {
    int             reason; /* Read-only */
    XEvent          *event; /* Read-only */
    String          *params;/* Read-only */
    Cardinal        nparams;/* Read-only */
    int             rotation;
    int             inclination;
    Boolean         doit;
} XrtRotateCallbackStruct;
```

reason is always XRT_REASON_ROTATE; *rotation* and *inclination* are the proposed values for XtNxrtGraphRotation and XtNxrtGraphInclination (these values can be modified by your callback procedure); *doit* is a flag that determines whether to allow the interaction to take place. To disable rotation, set *doit* to False in your callback procedure.

XtNxrtSetLabels **char **** **NULL** **CSG**
XtNxrtSetLabels2 **char **** **NULL** **CSG**

Specifies the string to render against the associated set value. Set-labels are displayed in the graph's legend. To clear all Set-labels, set XtNxrtSetLabels to NULL.

XtNxrtSetLabels2 is used to access the second graph's Set-labels in a combination graph.

XtNxrtTimeBase **time_t** **dynamic** **CSG**

Specifies the offset of time data since Jan. 1, 1970 GMT, the minimum allowed value of this resource. The default value is Jan. 1, 1970 in your time zone. For example, if your time zone is EST, this resource defaults to 18000 (since EST is 5 hours west of Greenwich). This is used whenever XtNxrtXAnnotationMethod is XRT_ANNO_TIME_LABELS.

time_t is defined in *<time.h>* and is usually typedef'd to a *long*. The ANSI C *mktime()* procedure is helpful when calculating suitable values for this resource.

XtNxrtTimeFormat **char *** **dynamic** **CSG**

Specifies the format string for Time-axis labels. This resource is only used when XtNxrtXAnnotationMethod is XRT_ANNO_TIME_LABELS. XRT/graph uses the string specified by this resource with your system's POSIX and ANSI C standard function *strftime()*. Please check

your system documentation for complete details on the formatting options. The following is a summary of ANSI formatting options:

%%	same as the "percent" character (%)
%a	locale's abbreviated weekday name
%A	locale's full weekday name
%b	locale's abbreviated month name
%B	locale's full month name
%c	locale's appropriate date and time representation
%d	ay of month (01 - 31)
%H	hour (00 - 23)
%I	hour (01 - 12)
%j	day number of year (001 - 366)
%m	month number (01 - 12)
%M	minute (00 - 59)
%p	locale's equivalent of either AM or PM
%S	seconds (00 - 61), allows for leap seconds
%U	week number of year (00 - 53), Sunday is the first day of week 1
%w	weekday number (0 - 6), Sunday = 0
%W	week number of year (00 - 53), Monday is the first day of week 1
%x	locale's appropriate date representation
%X	locale's appropriate time representation
%y	year within century (00 - 99)
%Y	year as ccyy (for example 1986)
%Z	time zone name or no characters if no time zone exists

When `NULL`, XRT/graph will generate a default format based on the range of data and the value of `XtNxrtTimeUnit`.

XtNxrtTimeFormatUseDefault **Boolean** **True** **CSG**

When `True`, XRT/graph will calculate a default time format dynamically. When an explicit time format is provided, XRT/graph sets this `Boolean` to `False`.

XtNxrtTimeUnit **XrtTimeUnit (enum)** **XRT_TMUNIT_SECONDS** **CSG**

Use this resource to tell XRT/graph how to interpret the X-values in data attached to the graph when `XtNxrtXAnnotationMethod` is `XRT_ANNO_TIME_LABELS`. Valid values are `XRT_TMUNIT_SECONDS`, `XRT_TMUNIT_MINUTES`, `XRT_TMUNIT_HOURS`, `XRT_TMUNIT_DAYS`, `XRT_TMUNIT_WEEKS`, `XRT_TMUNIT_MONTHS` and `XRT_TMUNIT_YEARS`.

For example, if data with X-values ranging from 0 to 4 has been attached to the graph, and if `XtNxrtTimeUnit` is set to `XRT_TMUNIT_WEEKS`, XRT/graph will annotate the X-axis with the four week period of time, beginning at the time specified by `XtNxrtTimeBase`.

XtNxrtTransformCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when XRT/graph is about to translate, zoom, or scale a graph. Transform callbacks are triggered by the *TranslateUpdate()*, *ZoomEnd()*, and *ViewReset()* actions. A pointer to an `XrtTransformCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                  graph,
    caddr_t                 client_data,
    XrtTransformCallbackStruct*call_data
    )
```

where `XrtTransformCallbackStruct` is defined as:

```
typedef struct {
    int             reason;     /* Read-only */
    XEvent          *event;     /* Read-only */
    String          *params;    /* Read-only */
    Cardinal        nparams;    /* Read-only */
    Boolean         reset;      /* Read-only */
    short           left_margin;
    short           right_margin;
    short           top_margin;
    short           bottom_margin;
    Boolean         doit;
} XrtTransformCallbackStruct;
```

reason is always `XRT_REASON_TRANSFORM`; *reset* notifies whether all scaling, translation, and zooming will be reset to XRT/graph defaults; *left_margin*, *right_margin*, *top_margin*, and *bottom_margin* are the proposed values for the Margin resources (these values can be modified by your callback procedure); *doit* is a flag that determines whether to allow the interaction to take place. To disable scaling and translation, set *doit* to `False` in your callback procedure.

XtNxrtType **XrtType (enum)** **XRT_TYPE_PLOT** **CSG**
XtNxrtType2 **XrtType (enum)** **XRT_TYPE_BAR** **CSG**

Determines the graph type. `XtNxrtType2` determines the graph type for the second graph in a combination graph. Valid values are `XRT_TYPE_PLOT`, `XRT_TYPE_AREA`, `XRT_TYPE_BAR`, `XRT_TYPE_STACKING_BAR`, `XRT_TYPE_PIE`, `XRT_TYPE_HILO`, `XRT_TYPE_HILO_OPEN_CLOSE`, `XRT_TYPE_CANDLE`, `XRT_TYPE_POLAR`, `XRT_TYPE_RADAR`, and `XRT_TYPE_FILLED_RADAR`.

XtNxrtXAnnoPlacement **XrtAnnoPlacement (enum)** **XRT_ANNO_AUTO** **CSG**
XtNxrtYAnnoPlacement **XrtAnnoPlacement (enum)** **XRT_ANNO_AUTO** **CSG**
XtNxrtY2AnnoPlacement **XrtAnnoPlacement (enum)** **XRT_ANNO_AUTO** **CSG**

Specifies the placement of the axis annotation and title with respect to the perpendicular axis. When `XRT_ANNO_AUTO`, the annotation and title are placed at the origin (or at the end of the axis closest to the origin for bar, HiLo, HiLoOpenClose, and candle graphs). These resources are ignored on polar, radar, and filled radar charts. `XtNxrtYAnnoPlacement` is ignored on dual-axis graphs. `XtNxrtY2AnnoPlacement` is ignored unless the Y2-axis is the primary Y-axis.

Other values are: `XRT_ANNO_ORIGIN`, which places the annotation/title along the origin; `XRT_ANNO_MIN`, which places annotation/title at the axis minimum; and `XRT_ANNO_MAX`, which places annotation/title at the axis maximum.

XtNxrtXAnnotationMethod	**XrtAnnoMethod (enum)XRT_ANNO_VALUES**		**CSG**
XtNxrtYAnnotationMethod	**XrtAnnoMethod (enum)XRT_ANNO_VALUES**		**CSG**
XtNxrtY2AnnotationMethod	**XrtAnnoMethod (enum)XRT_ANNO_VALUES**		**CSG**

Specifies the method used to annotate the axes. The following methods are valid for the X-axis: `XRT_ANNO_VALUES`, `XRT_ANNO_POINT_LABELS`, `XRT_ANNO_VALUE_LABELS`, and `XRT_ANNO_TIME_LABELS`. For the Y- and Y2-axes, only `XRT_ANNO_VALUES` and `XRT_ANNO_VALUE_LABELS` are valid; attempting to set to an invalid value results in `XRT_ANNO_VALUES` being used.

- When set to `XRT_ANNO_VALUES`, XRT/graph annotates the axis at appropriate intervals along the axis using the graph's X- or Y-values and the values of `XtNxrt[XYY2]NumMethod`, `XtNxrt[XYY2]Num`, `XtNxrt[XYY2]Tick`, and `XtNxrt[XYY2]Precision`. Values may be formatted using `XtNxrt[XYY2]AxisLabelFormat`.
- When set to `XRT_ANNO_VALUE_LABELS`, the value of `XtNxrt[XYY2]Labels` is used to annotate the axis. Not allowed on the X-axis of radar or filled radar charts.
- When set to `XRT_ANNO_POINT_LABELS` for the X-axis, the value of `XtNxrtPointLabels` (or `XtNxrtPointLabels2`) is used to annotate the X-axis. Note that Point-labels are usually not meaningful in graphs using General data.
- When set to `XRT_ANNO_TIME_LABELS` for the X-axis, Time-labels are used to annotate the X-axis. Not allowed on polar, radar, or filled radar charts.

XtNxrtXAnnotationRotation	**XrtRotate (enum)**	**XRT_ROTATE_NONE**	**CSG**
XtNxrtYAnnotationRotation	**XrtRotate (enum)**	**XRT_ROTATE_NONE**	**CSG**
XtNxrtY2AnnotationRotation	**XrtRotate (enum)**	**XRT_ROTATE_NONE**	**CSG**

Use this resource to rotate axis annotation text. When set to `XRT_ROTATE_90` or `XRT_ROTATE_270`, axis annotations are rotated either 90 or 270 degrees counterclockwise.

Using rotated text can slow down the real-time performance of the graph widget dramatically, and is therefore not recommended for graphs that may need to be updated many times per second.

`XtNxrtXAnnotationMethod` is ignored on polar, radar, and filled radar charts.

XtNxrtXAxisDataStyle	**XrtDataStyle ***	**solid black line of width 1**	**CSG**
XtNxrtYAxisDataStyle	**XrtDataStyle ***	**solid black line of width 1**	**CSG**
XtNxrtY2AxisDataStyle	**XrtDataStyle ***	**solid black line of width 1**	**CSG**

Specifies the `XrtDataStyle` to be used when rendering the particular axis. In particular, the point size (which the tick length is based on), line style, color and width elements of the `XrtDataStyle` structure are used. If not specified, XRT/graph will use a 1-pixel wide solid line in the graph foreground color (which defaults to black).

A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrtXAxisDataStyleUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYAxisDataStyleUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2AxisDataStyleUseDefault	**Boolean**	**True**	**CSG**

When `True`, the default axis data style will be used. When explicit data styles are provided, XRT/graph sets this Boolean to `False`.

XtNxrtXAxisLabelFormat **char *** **NULL** **CSG**
XtNxrtYAxisLabelFormat **char *** **NULL** **CSG**
XtNxrtY2AxisLabelFormat **char *** **NULL** **CSG**

Specifies how to format axis annotation, used when the annotation method is `XRT_ANNO_VALUES`. When set to `NULL`, annotation is not formatted. See 4.4 on page 66 for complete details on using axis label formatting. Valid format codes are:

`#  0  ?`	Value placeholders. Used with . (period) to fix number of digits. 0 displays zero when no digit exits in the value. ? works like 0 and adds space for insignificant zeros to align decimals and fractions.
. (period)	Inserts decimal point.
, (comma)	Thousands separator, or scales value if at end.
`$  -  +  /  (  )`	Inserts character at that point in the value.
`\<`*char*`>`	Inserts *<char>* at that point in the value.
`_<`*char*`>`	Inserts space instead of *<char>*.
`%`	Multiplies value by 100 and displays % sign.
`[multiplier` *n*`]`	Multiplies value by *n*.
`E+  e+  E-  E-`	Converts value to scientific notation. Must be followed by value placeholders to specify number of digits in exponent.
`*<`*char*`>`	Pads annotation space with *<char>*.
`[hide] [show]`	Hides or shows value/formats that follow.
; (semicolon)	Ends formatting commands for one group and begins another group. First three groups are positive, negative, and zero values.
`[<val] [>val] [=val] [<>val] [>=val] [<=val]`	Defines conditional formatting group. If *val* meets the condition, the codes that follow are applied to the value.

XtNxrtXAxisLogarithmic **Boolean** **False** **CSG**
XtNxrtYAxisLogarithmic **Boolean** **False** **CSG**
XtNxrtY2AxisLogarithmic **Boolean** **False** **CSG**

When `True`, XRT/graph graphs the data against a logarithmic axis. All data values less than or equal to zero will be displayed as holes (missing data values). It is not possible to set `XtNxrt[XYY2]Tick`, `XtNxrt[XYY2]Num` or `XtNxrt[XYY2]Precision` for a logarithmic axis. Since logarithmic axes can only graph values greater than zero, setting `XtNxrt[XYY2]AxisMin`, `XtNxrt[XYY2]AxisMax`, `XtNxrt[XYY2]Min`, `XtNxrt[XYY2]Max` or `XtNxrt[XY]Origin` less than or equal to zero forces its corresponding `UseDefault` resource to `True`.

`XtNxrtXAxisLogarithmic` is ignored on bar, stacking bar, polar, radar, or filled radar charts; also when the X-axis annotation method is `XRT_ANNO_TIME_LABELS`. All logarithmic resources are ignored on pie charts.

XtNxrtXAxisMax	**float**	**dynamic**	**CSG**
XtNxrtYAxisMax	**float**	**dynamic**	**CSG**
XtNxrtY2AxisMax	**float**	**dynamic**	**CSG**

Specifies the maximum value of the axis. Data is clipped to the smaller of `XtNxrt[XYY2]AxisMax` and `XtNxrt[XYY2]Max`. A side-effect of setting this resource is that the corresponding `UseDefault` resource is set `False`.

Unless set explicitly, the value of this resource is derived from the values of `XtNxrt[XYY2]Max`, `XtNxrt[XY]Origin`, `XtNxrt[XYY2]Num` and `XtNxrt[XYY2]NumMethod`, and falls on an integral multiple of `XtNxrt[XYY2]Num`.

NOTE: When its corresponding `UseDefault` resource is `False`, any change in the axis resources that makes `XtNxrt[XYY2]AxisMax` invalid causes XRT/graph to set the `UseDefault` resource to `True`. This forces the axis bounds to be recalculated, which deals with all invalid cases.

`XtNxrt[XYY2]AxisMax` is ignored in pie, polar, radar, and filled radar charts. If a relationship exists between the first and second Y-axis, the default for `XtNxrtY2AxisMax` is calculated from `XtNxrtYAxisMax`.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXAxisMaxUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYAxisMaxUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2AxisMaxUseDefault	**Boolean**	**True**	**CSG**

When `True`, XRT/graph will calculate the maximum axis value. When you explicitly set `XtNxrt[XYY2]AxisMax`, this resource is set to `False`. You cannot set this resource to `False` unless a value for `XtNxrt[XYY2]AxisMax` has been provided or calculated.

NOTE: When `False`, any change in the axis resources that makes `XtNxrt[XYY2]AxisMax` invalid causes XRT/graph to set the `UseDefault` resource to `True`, forcing the axis bounds to be recalculated.

XtNxrtXAxisMin	**float**	**dynamic**	**CSG**
XtNxrtYAxisMin	**float**	**dynamic**	**CSG**
XtNxrtY2AxisMin	**float**	**dynamic**	**CSG**

Specifies the minimum value of the axis. Graph data is clipped to the larger of `XtNxrt[XYY2]AxisMin` and `XtNxrt[XYY2]Min`. A side-effect of setting this resource is that the corresponding `UseDefault` resource is set to `False`.

Unless set explicitly, the value of this resource is derived from the values of `XtNxrt[XYY2]Min`, `XtNxrt[XY]Origin`, `XtNxrt[XYY2]Num` and `XtNxrt[XYY2]NumMethod` resources, and falls on an integral multiple of `XtNxrt[XYY2]Num`.

NOTE: When its corresponding `UseDefault` resource is `False`, any change in the axis resources that makes `XtNxrt[XYY2]AxisMin` invalid causes XRT/graph to set the `UseDefault` resource to `True`. This forces the axis bounds to be recalculated, which deals with all invalid cases.

`XtNxrt[XYY2]AxisMin` is ignored in pie, polar, radar, and filled radar charts. If a relationship exists between the first and second Y-axis, the default for `XtNxrtY2AxisMin` is calculated from `XtNxrtYAxisMin`.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXAxisMinUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYAxisMinUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2AxisMinUseDefault	**Boolean**	**True**	**CSG**

When `True`, XRT/graph will calculate the minimum axis value. When you explicitly set `XtNxrt[XYY2]AxisMin`, this resource is set to `False`. You cannot set this resource to `False` unless a value for `XtNxrt[XYY2]AxisMin` has been provided or calculated.

NOTE: When `False`, any change in the axis resources that makes `XtNxrt[XYY2]AxisMin` invalid causes XRT/graph to set the `UseDefault` resource to `True`, forcing the axis bounds to be recalculated.

XtNxrtXAxisReversed	**Boolean**	**False**	**CSG**
XtNxrtYAxisReversed	**Boolean**	**False**	**CSG**
XtNxrtY2AxisReversed	**Boolean**	**False**	**CSG**

When `True`, XRT/graph will draw and annotate the X-, Y-, or Y2-axis in the reverse direction. This is commonly used to cause the X-axis annotation to increase from right to left or the Y-axis annotation to increase downward.

`XtNxrtY2AxisReversed` is ignored when `XtNxrtYAxisMult` is non-zero. If a positive multiplier is specified, the second Y-axis will have the same direction as the first Y-axis. If a negative multiplier is specified, the second Y-axis will have the opposite direction to the first.

XtNxrtXAxisShow	**Boolean**	**True**	**CSG**
XtNxrtYAxisShow	**Boolean**	**True**	**CSG**
XtNxrtY2AxisShow	**Boolean**	**False**	**CSG**

Determines if the axis should be displayed.

All Y2-axis resources are ignored unless a second dataset has been attached (using `XtNxrtData2`) or `XtNxrtYAxisMult` is non-zero.

XtNxrtXGrid	**float**	**0.0**	**CSG**
XtNxrtYGrid	**float**	**0.0**	**CSG**
XtNxrtY2Grid	**float**	**0.0**	**CSG**

Specifies the increment between gridlines, starting from the origin. `XtNxrtY2Grid` is ignored unless the Y2-axis is the primary Y-axis. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

No gridlines are displayed when the value is 0.0.

When `XtNxrt[XYY2]GridUseDefault` is set to `True`, the default grid increment is the value of `XtNxrt[XYY2]Num`. Gridlines are drawn through each axis label.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXGridUseDefault	**Boolean**	**False**	**CSG**
XtNxrtYGridUseDefault	**Boolean**	**False**	**CSG**
XtNxrtY2GridUseDefault	**Boolean**	**False**	**CSG**

When `True`, XRT/graph determines grid increments at render-time. When you explicitly set `XtNxrt[XYY2]Grid`, these resources are set to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Grid` values.

When `XtNxrt[XYY2]GridUseDefault` is set to `True`, the default grid increment is the value of `XtNxrt[XYY2]Num` and gridlines are drawn through each axis label.

XtNxrtXGridDataStyle	**XrtDataStyle ***	**dotted line**	**CSG**
XtNxrtYGridDataStyle	**XrtDataStyle ***	**dotted line**	**CSG**
XtNxrtY2GridDataStyle	**XrtDataStyle ***	**dotted line**	**CSG**

Specifies the `XrtDataStyle` to be used when rendering the X and Y gridlines. In particular, line style, color and width elements of the `XrtDataStyle` structure are used. If not specified, XRT/graph will use a 1-pixel wide dotted line in the graph foreground color.

A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`. `XtNxrtY2GridDataStyle` is ignored unless the Y2-axis is the primary Y-axis.

XtNxrtXGridDataStyleUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYGridDataStyleUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2GridDataStyleUseDefault	**Boolean**	**True**	**CSG**

When `True`, the default grid data style will be used. When explicit data styles are provided, XRT/graph sets this Boolean to `False`.

XtNxrtXLabels	**XrtValueLabel ****	**NULL**	**CSG**
XtNxrtYLabels	**XrtValueLabel ****	**NULL**	**CSG**
XtNxrtY2Labels	**XrtValueLabel ****	**NULL**	**CSG**

These resources supply the labels for the X-, Y-, and Y2-axis when the value of `XtNxrt[XYY2]AnnotationMethod` is `XRT_ANNO_VALUE_LABELS`. The resource value is a `NULL`-terminated array of pointers to `XrtValueLabel` structures. Each structure contains a Value-label pair.

Value-labels cannot be used to annotate pie charts or the X-axis of radar or filled radar charts.

When used in an *XtGetValues()* call, a `NULL` pointer is returned if there are no Value-labels defined. The returned Value-labels are sorted by their X- or Y-value.

XtNxrtXMarker	**float**	**0.0**	**CSG**
XtNxrtYMarker	**float**	**0.0**	**CSG**

Specifies position of the markers in graph coordinates. A marker will not be displayed unless the corresponding `MarkerShow` resource is `True`, and, for X-markers, the `XtNxrtXMarkerMethod` resource is set to `XRT_MARKER_VALUE`.

When positioning markers on combination graphs, set `XtNxrtMarkerDataset` appropriately before using these resources.

`XtNxrt[XY]Marker` is ignored on pie, radar, and filled radar charts. `XtNxrtYMarker` is also ignored on polar charts.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXMarkerDataStyle	**XrtDataStyle ***	**dynamic**	**CSG**
XtNxrtYMarkerDataStyle	**XrtDataStyle ***	**dynamic**	**CSG**

Specifies the `XrtDataStyle` to be used when for an individual X- or Y-marker. In particular, line style, color and width elements of the `XrtDataStyle` structure are used. If never specified, XRT/graph uses the value of `XtNxrtMarkerDataStyle` for the markers (its default is a black, 1-pixel wide dashed line). When `XtNxrt[XY]MarkerDataStyle` is explicitly set, it is used instead of `XtNxrtMarkerDataStyle` for the marker.

XtNxrtXMarkerDataStyleUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYMarkerDataStyleUseDefault	**Boolean**	**True**	**CSG**

When `True`, the default marker data style will be used. When explicit data styles values are provided, XRT/graph sets this resource to `False`.

XtNxrtXMarkerMethod	**XrtMarkerMethod**	**XRT_MARKER_SET_POINT**	**CSG**

Specifies which resources are used to position the X-marker. When set to `XRT_MARKER_SET_POINT`, the `XtNxrtXMarkerSet` and `XtNxrtXMarkerPoint` resources are used. When set to `XRT_MARKER_VALUE`, the `XtNxrtXMarker` resource is used.

Pie, radar, and filled radar charts can only use set-point X-marker positioning.

XtNxrtXMarkerPoint	**int**	**0**	**CSG**
XtNxrtXMarkerSet	**int**	**0**	**CSG**

Specifies the set and point indices through which to draw the X-marker. This is the only way to specify a marker on pie, radar, and filled radar charts. The marker will not be displayed unless the corresponding `MarkerShow` resource is `True` and the `XtNxrtXMarkerMethod` resource is set to `XRT_MARKER_SET_POINT`.

If the Y-value of the point is the hole value, no marker is drawn. When positioning markers on combination graphs, set `XtNxrtMarkerDataset` appropriately before using these resources.

XtNxrtXMarkerShow	**Boolean**	**False**	**CSG**
XtNxrtYMarkerShow	**Boolean**	**False**	**CSG**

Specifies whether to display the corresponding marker. The `XtNxrt[XY]Marker`, `XtNxrtXMarkerSet` and `XtNxrtXMarkerPoint` resources position the markers and `XtNxrtXMarkerMethod` chooses the X-marker method.

XtNxrtXMax	**float**	**dynamic**	**CSG**
XtNxrtYMax	**float**	**dynamic**	**CSG**
XtNxrtY2Max	**float**	**dynamic**	**CSG**

Specifies the value of the high end of the X, Y and Y2 data. Data values greater than this value will not be displayed. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

`XtNxrt[XYY2]Max` is ignored in pie charts. In polar, radar, and filled radar charts, setting `XtNxrtXMax` has no effect, and `XtNxrtYMax` cannot be set to a value less than the maximum Y-value in the data.

If a relationship exists between the first and second Y-axis, then the default for `XtNxrtY2Max` is calculated from `XtNxrtYMax`.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXMaxUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYMaxUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2MaxUseDefault	**Boolean**	**True**	**CSG**

When `True`, `XtNxrt[XYY2]Max` values are determined by XRT/graph at render-time. When explicit `XtNxrt[XYY2]Max` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Max` values.

XtNxrtXMin	**float**	**dynamic**	**CSG**
XtNxrtYMin	**float**	**dynamic**	**CSG**
XtNxrtY2Min	**float**	**dynamic**	**CSG**

Specifies the value of the low end of the X, Y and Y2 data. Data values less than this value will not be displayed. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

`XtNxrt[XYY2]Min` is ignored in pie charts. In polar, radar, and filled radar charts, setting `XtNxrtXMin` has no effect, and `XtNxrtYMin` cannot be set to a value greater than the minimum Y-value in the data.

If a relationship exists between the first and second Y-axis, then the default for `XtNxrtY2Min` is calculated from `XtNxrtYMin`.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXMinUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYMinUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2MinUseDefault	**Boolean**	**True**	**CSG**

When `True`, `XtNxrt[XYY2]Min` values are determined by XRT/graph at render-time. When explicit `XtNxrt[XYY2]Min` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Min` values.

XtNxrtXNum	**float**	**dynamic**	**CSG**
XtNxrtYNum	**float**	**dynamic**	**CSG**
XtNxrtY2Num	**float**	**dynamic**	**CSG**

Specifies the increment between axis-numbering. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

`XtNxrt[XYY2]Num` is ignored when the axis is logarithmic, and in pie charts.

The default value `XtNxrt[XYY2]Num` is derived from the data unless the corresponding `XtNxrt[XYY2]Tick` has been explicitly set. In this case, the numbering increment will default to twice the tick increment.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXNumUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYNumUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2NumUseDefault	**Boolean**	**True**	**CSG**

When `True`, `XtNxrt[XYY2]Num` values are determined by XRT/graph at render-time. When explicit `XtNxrt[XYY2]Num` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Num` values.

XtNxrtXNumMethod	**XrtNumMethod (enum)XRT_NUM_ROUND**		**CSG**
XtNxrtYNumMethod	**XrtNumMethod (enum)XRT_NUM_ROUND**		**CSG**
XtNxrtY2NumMethod	**XrtNumMethod (enum)XRT_NUM_ROUND**		**CSG**

Specifies the method used to calculate axis numbering. These resources control the default value of the `XtNxrt[XYY2]Num` resources. If the value is `XRT_NUM_PRECISION`, the current value of the axis precision (`XtNxrt[XYY2]Precision`) is used to derive the value of `XtNxrt[XYY2]Num`. If the value is `XRT_NUM_ROUND`, then rounded numbers are chosen (most significant digit is 1, 2 or 5). In either case, the current value of the axis precision is used to format the numbers.

XtNxrtXOrigin	**float**	**dynamic**	**CSG**
XtNxrtYOrigin	**float**	**dynamic**	**CSG**
XtNxrtY2Origin	**float**	**dynamic**	**CSG**

Specifies where the axis should be rendered. For example, setting X origin to 5.0 will cause the Y-axis to cross the X-axis at X=5.0. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

`XtNxrt[XYY2]Origin` is ignored in pie charts. `XtNxrtY2Origin` is ignored unless the Y2-axis is the primary Y-axis.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXOriginBase	**float**	**0.0**	**CSG**

Specifies the angle to use as the X-axis origin for polar, radar, and filled radar charts. All angles are referenced from this angle, treating it as zero. The value of this resource is interpreted in the units specified by `XtNxrtAngleUnit`.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXOriginPlacement	**XrtOriginPlacement (enum)XRT_ORIGIN_AUTO**		**CSG**
XtNxrtYOriginPlacement	**XrtOriginPlacement (enum)XRT_ORIGIN_AUTO**		**CSG**
XtNxrtY2OriginPlacement	**XrtOriginPlacement (enum)XRT_ORIGIN_AUTO**		**CSG**

Specifies the location of the origin. When `XRT_ORIGIN_AUTO`, the origin is placed as follows:

- Plot/bubble/area graphs – origin placed at the axis minimum or at zero if the dataset has both positive and negative values.
- Bar, HiLo, HiLoOpenClose, candle charts – origin placed at zero.

Other values are: `XRT_ORIGIN_ZERO`, which places the origin at zero; `XRT_ORIGIN_MIN`, which places the origin at the axis minimum; and `XRT_ORIGIN_MAX`, which places the origin at the axis maximum.

Stacking bar charts always have their origin at zero; attempts to place it elsewhere are ignored. These resources are ignored in pie charts. `XtNxrtY2OriginPlacement` is ignored unless the Y2-axis is the primary Y-axis.

This resource is ignored if you set an explicit origin with `XtNxrt[XYY2]Origin`.

XtNxrtXOriginUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYOriginUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2OriginUseDefault	**Boolean**	**True**	**CSG**

When `True`, XRT/graph calculates the origin values at render-time. When explicit `XtNxrt[XYY2]Origin` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Origin` values.

XtNxrtXPrecision	**int**	**dynamic**	**CSG**
XtNxrtYPrecision	**int**	**dynamic**	**CSG**
XtNxrtY2Precision	**int**	**dynamic**	**CSG**

This resource defines the number of digits of precision after the decimal point to be used for axis-numbering. If 0 or less, axis-numbers will all be integers. XRT/graph will also use the precision to determine working values for the axis' origin, min, max, num and tick increments. If, for example, `XtNxrtXOrigin` is set to 1.45 and `XtNxrtXPrecision` is 0, XRT/graph will draw the Y-axis at X=1. If the `XtNxrtXPrecision` is subsequently changed to 2, XRT/graph will draw the Y-axis at X=1.45.

Negative precision values indicates precision in powers of 10. For example, setting `XtNxrtYPrecision` to -2 will cause all Y-axis numbering to be multiples of 100.

A side effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrtXPrecisionUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYPrecisionUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2PrecisionUseDefault	**Boolean**	**True**	**CSG**

When `True`, XRT/graph calculates appropriate values for `XtNxrt[XYY2]Precision` automatically at render-time. When explicit `XtNxrt[XYY2]Precision` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Precision` values.

XtNxrtXTick	**float**	**dynamic**	**CSG**
XtNxrtYTick	**float**	**dynamic**	**CSG**
XtNxrtY2Tick	**float**	**dynamic**	**CSG**

Specifies the increment between axis-ticks. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

The default `XtNxrt[XYY2]Tick` value is derived from the data unless the corresponding `XtNxrt[XYY2]Num` has been explicitly set. In this case, the tick increment will default to half the numbering increment.

`XtNxrt[XYY2]Tick` is ignored in pie charts or if the axis is logarithmic.

When the X-axis annotation method is Point-labels, Value-labels, or Time-labels, and `XtNxrtXTickUseDefault` is `True`, ticks will display at each label. Otherwise `XtNxrtXTickUseDefault` is `False`, so the specified X tick increment will be honored. If the increment is zero, no ticks will display.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtXTickUseDefault	**Boolean**	**True**	**CSG**
XtNxrtYTickUseDefault	**Boolean**	**True**	**CSG**
XtNxrtY2TickUseDefault	**Boolean**	**True**	**CSG**

When `True`, `XtNxrt[XYY2]Tick` values are determined by XRT/graph at render-time. When explicit `XtNxrt[XYY2]Tick` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[XYY2]Tick` values.

XtNxrtXTitle	**char ***	**NULL**	**CSG**
XtNxrtYTitle	**char ***	**NULL**	**CSG**
XtNxrtY2Title	**char ***	**NULL**	**CSG**

Specifies the title to display on the X-, Y- and Y2-axis. These resources are ignored on pie, polar, radar, or filled radar charts.

XtNxrtXTitleRotation	**XrtRotate (enum)**	**XRT_ROTATE_NONE**	**CSG**
XtNxrtYTitleRotation	**XrtRotate (enum)**	**XRT_ROTATE_NONE**	**CSG**
XtNxrtY2TitleRotation	**XrtRotate (enum)**	**XRT_ROTATE_NONE**	**CSG**

Use this resource to rotate the title text. It can only be applied to vertically-oriented axes (Y or Y2 unless `XtNxrtInvertOrientation` is `True`). When set to `XRT_ROTATE_NONE`, the title is centered above or below the axis. When set to `XRT_ROTATE_90` or `XRT_ROTATE_270`, the title is rotated either 90 or 270 degrees counterclockwise and is centered to the left or right of the vertical axis.

Using rotated text can slow down the real-time performance of the graph widget dramatically, and therefore is not recommended for graphs that need to be updated many times per second.

XtNxrtYAnnotationAngle **float** **dynamic** **CSG**
XtNxrtY2AnnotationAngle **float** **dynamic** **CSG**

Specifies the angle to draw Y- and Y2-axis annotation. Only used for polar, radar and filled radar plots. The default is the current X-origin value (specified by `XtNxrtXOrigin`). `XtNxrtY2AnnotationAngle` is only used when the Y2-axis is the primary Y-axis.

A side effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtYAnnotationAngleUseDefault **Boolean** **True** **CSG**
XtNxrtY2AnnotationAngleUseDefault **Boolean** **True** **CSG**

When `True`, XRT/graph determines appropriate values for `XtNxrt[YY2]AnnotationAngle` automatically at render-time. When explicit `XtNxrt[YY2]AnnotationAngle` values are provided, XRT/graph sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt[YY2]AnnotationAngle` values.

XtNxrtYAxis100Percent **Boolean** **False** **CSG**
XtNxrtY2Axis100Percent **Boolean** **False** **CSG**

Specifies whether to use a percentage interpretation for the Y- or Y2-axis annotation of a chart which is stacked. When set to `True`, the data is interpreted as a percentage of the total of the Y-values in the graph. This total represents 100%. When `False`, no percentage interpretation occurs.

These resources are only used when the graph type is `XRT_TYPE_STACKING_BAR` or a type for which a graph can be stacked and if `XtNxrtIsStacked` or `XtNxrtIsStacked2` is `True`. See page 182 for applicable graph types.

XtNxrtYAxisConst **float** **0.0** **CSG**

Used in conjunction with `XtNxrtYAxisMult` to specify a relationship between the first and second Y-axis. To indicate that no relationship exists between the two Y-axes, set `XtNxrtYAxisMult` to 0.0.

For example, if the first Y-axis is degrees Celsius, and the second Y-axis is degrees Fahrenheit, set `XtNxrtYAxisMult` to 9.0/5.0, and set `XtNxrtYAxisConst` to 32.0.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtYAxisMult **float** **0.0** **CSG**

Used in conjunction with `XtNxrtYAxisConst` to specify a relationship between the first and second Y-axes. To indicate that no relationship exists between the two Y-axes, set `XtNxrtYAxisMult` to 0.0.

For example, if the first Y-axis is degrees Celsius, and the second Y-axis is degrees Fahrenheit, set `XtNxrtYAxisMult` to 9.0/5.0, and set `XtNxrtYAxisConst` to 32.0.

Remember to use *XrtFloatToArgVal()* when setting resources of type *float*.

XtNxrtZoomAxisCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when XRT/graph is about to zoom into graph by changing the axis bounds. ZoomAxis callbacks are triggered by the

ZoomAxisEnd() action. A pointer to an `XrtZoomAxisCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget                      graph,
    caddr_t                     client_data,
    XrtZoomAxisCallbackStruct   *call_data
    )
```

where `XrtZoomAxisCallbackStruct` is defined as:

```
typedef struct {
    int         reason;     // Read-only
    XEvent      *event;     // Read-only
    String      *params;    // Read-only
    Cardinal    nparams;    // Read-only
    Boolean     reset;      // Read-only
    float       new_xaxis_min;
    float       new_xaxis_max;
    float       new_yaxis_min;
    float       new_yaxis_max;
    float       new_y2axis_min;
    float       new_y2axis_max;
    Boolean     doit;
} XrtZoomAxisCallbackStruct;
```

reason is always `XRT_REASON_ZOOM_AXIS`; *reset* notifies whether all axis zooming will be reset to XRT/graph defaults; *new_xaxis_min* through *new_y2axis_max* are the proposed values for the Axes minimum and maximum resources (these values can be modified by your callback procedure); *doit* is a flag that determines whether to allow the interaction to take place. To disable axis zooming, set *doit* to `False` in your callback procedure.

A.3 Inherited Resources

XmPrimitive Resource Set			
Name	**Type**	**Default**	**Access**
XmNhighlightColor	Pixel	dynamic	CSG
XmNhighlightOnEnter	Boolean	False	CSG
XmNhighlightPixmap	Pixmap	dynamic	CSG
XmNnavigationType	XmNavigationType	XmNONE	CSG
XmNshadowThickness	Dimension	2	CSG
XmNtraversalOn	Boolean	True	CSG
XmNuserData	XtPointer	NULL	CSG

Core Resource Set			
Name	**Type**	**Default**	**Access**
XmNdestroyCallback	XtCallbackList	NULL	C
XmNheight	Dimension	dynamic	CSG
XmNtranslations	XtTranslations	see Appendix E on page 251	CSG
XmNwidth	Dimension	dynamic	CSG
XmNx	Position	0	CSG
XmNy	Position	0	CSG

B

Text Object Attribute Reference

This appendix lists all of the TextArea object attributes in alphabetical order. Listed after the attribute name is its data type and default value.

Instantiation: `hTextObject = XrtTextAreaCreate(graph)`

Attribute Methods: *XrtTextAreaGetValues()*, *XrtTextAreaSetValues()*

XRT_TEXT_ADJUST **XrtAdjust (enum)** **XRT_ADJUST_CENTER**

Specifies how multiple lines of text should be aligned within the text object. Valid values are `XRT_ADJUST_LEFT`, `XRT_ADJUST_CENTER` and `XRT_ADJUST_RIGHT`.

XRT_TEXT_ANCHOR **XrtAnchor (enum)** **XRT_ANCHOR_BEST**

Specifies where to position the text object relative to the attachment point. Valid values are: `XRT_ANCHOR_NORTH`, `XRT_ANCHOR_SOUTH`, `XRT_ANCHOR_EAST`, `XRT_ANCHOR_WEST`, `XRT_ANCHOR_NORTHWEST`, `XRT_ANCHOR_NORTHEAST`, `XRT_ANCHOR_SOUTHWEST`, `XRT_ANCHOR_SOUTHEAST`, `XRT_ANCHOR_HOME`, and `XRT_ANCHOR_BEST`.

XRT_TEXT_ATTACH_DATASET **int** **XRT_DATASET1**

Specifies which dataset to use for determining the attachment point. This attribute is used with all attachment types except `XRT_TEXT_ATTACH_PIXEL`.

XRT_TEXT_ATTACH_PIXEL_X **Position** **0**
XRT_TEXT_ATTACH_PIXEL_Y **Position** **0**

Specifies the widget pixel coordinates to attach the text object to. These attributes are used only when `XRT_TEXT_ATTACH_TYPE` is set to `XRT_TEXT_ATTACH_PIXEL`.

XRT_TEXT_ATTACH_POINT **int** **0**
XRT_TEXT_ATTACH_SET **int** **0**

Specifies the data point (through its set and point index) to attach the text object to. These attributes are used when XRT_TEXT_ATTACH_TYPE is set to XRT_TEXT_ATTACH_DATA or XRT_TEXT_ATTACH_DATA_VALUE.

XRT_TEXT_ATTACH_VALUE_X **float** **0.0**
XRT_TEXT_ATTACH_VALUE_Y **float** **0.0**

Specifies the graph X- and Y-value to attach the text object to. These attributes are used when XRT_TEXT_ATTACH_TYPE is set to XRT_TEXT_ATTACH_VALUE. XRT_TEXT_ATTACH_VALUE_Y is also used when the attachment type is XRT_TEXT_ATTACH_DATA_VALUE.

XRT_TEXT_ATTACH_TYPE **XrtAttachType (enum)** **XRT_TEXT_ATTACH_VALUE**

Specifies how the text object is attached to the graph. Valid values are XRT_TEXT_ATTACH_PIXEL, XRT_TEXT_ATTACH_VALUE, XRT_TEXT_ATTACH_DATA and XRT_TEXT_ATTACH_DATA_VALUE. Each attachment type uses several other Attachment attributes to determine the actual positioning.

XRT_TEXT_BACKGROUND_COLOR **char *** **NULL**

Specifies the text object background color. When XRT_TEXT_BACKGROUND_COLOR and XRT_TEXT_IMAGE are both NULL, the text object is transparent. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green and, Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome Server.

XRT_TEXT_BORDER **XrtBorder (enum)** **XRT_BORDER_NONE**

Specifies the style of border used to identify the text object. Valid styles are XRT_BORDER_NONE, XRT_BORDER_PLAIN, XRT_BORDER_3D_IN, XRT_BORDER_3D_OUT, XRT_BORDER_ETCHED_IN, XRT_BORDER_ETCHED_OUT, XRT_BORDER_FRAME_IN, XRT_BORDER_FRAME_OUT, XRT_BORDER_BEVEL, and XRT_BORDER_SHADOW.

XRT_TEXT_BORDER_WIDTH **Dimension** **2**

Specifies the width of the text object border in pixels. Must be between 0 and 20.

XRT_TEXT_FONT **XmFontList** **current axis font**

Specifies the font used for the text contained in the text object.

XRT_TEXT_FOREGROUND_COLOR **char *** **NULL**

Specifies the text object foreground color. When NULL, the widget foreground color is used. The string is either a color name, or a '#' character followed by 2 hexadecimal characters for each of the Red, Green, and Blue components. See section 2.8 on page 29 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XRT_TEXT_IMAGE **XrtImage** **NULL**

Specifies the filename of an image to display in the background of the text object. See Background Images on page 91 for more information on setting background images.

XRT_TEXT_IMAGE_LAYOUT **XrtImageLayout** **XRT_IMAGELAYOUT_CENTERED**

Specifies the positioning and scaling of the background image. Valid values are: `XRT_IMAGELAYOUT_CENTERED` and `XRT_IMAGELAYOUT_TILED`. See Background Images on page 91 for more information on setting background images.

XRT_TEXT_IMAGE_MINIMUM_SIZE **Boolean** **False**

Specifies the minimum size for a label that corresponds to the size of the image. This is not available for non-text related images (for example, the graph area background image). See Background Images on page 91 for more information on setting background images.

XRT_TEXT_IS_CONNECTED **Boolean** **False**

Specifies whether to draw a connecting line from the text object to the point of attachment. This is a pixel-wide line drawn in the foreground color of the text object.

XRT_TEXT_IS_SHOWING **Boolean** **True**

Specifies whether to display or hide the text object.

XRT_TEXT_NAME **char *** **NULL**

Specifies the name of the text object. This is provided for application use and is not used by the widget.

XRT_TEXT_OFFSET **int** **0**

Specifies the pixel offset between the center of the text object and the point of attachment. The direction of the offset is the same as the anchor.

XRT_TEXT_PRINT_FONT **char *** **NULL**

Specifies the font used for the text object when printing the XRT/graph widget using *XrtDrawPS()*. When `NULL`, *XrtDrawPS()* automatically chooses the closest matching PostScript font;[1] This attribute can be set to any PostScript font.

XRT_TEXT_ROTATION **XrtRotate (enum)** **XRT_ROTATE_NONE**

Specifies the rotation of text displayed in the text object. When set to `XRT_ROTATE_90` or `XRT_ROTATE_270`, text is rotated either 90 or 270 degrees counterclockwise.

Using rotated text can slow down the real-time performance of the graph widget dramatically, and is therefore not recommended for graphs that may need to be updated many times per second.

1. PostScript fonts are chosen from the 35 PostScript fonts available on most PostScript printers (see Figure 92 on page 227 for the complete list).

XRT_TEXT_STRINGS **char **** **NULL**

Specifies a NULL-terminated list of strings displayed in the text object.

XRT_TEXT_WIDTH **Dimension** **dynamic**
XRT_TEXT_HEIGHT **Dimension** **dynamic**

Contains the current height and width of the text object in pixels.
NOTE: This attribute is read-only; text object height or width cannot be set.

XRT_TEXT_X **Position** **dynamic**
XRT_TEXT_Y **Position** **dynamic**

Contains the current (X,Y) coordinates of the top-left corner of the text object.
NOTE: This attribute is read-only; use the Attachment attributes to position a text object.

C

PointStyle Object Attribute Reference

This appendix lists all of the PointStyle object attributes in alphabetical order. Listed after the attribute name is its data type and default value.

Instantiation: `hPointStyleObject = XrtPointStyleCreate(graph)`

Attribute Methods: *XrtPointStyleGetValues()*, *XrtPointStyleSetValues()*

XRT_POINTSTYLE_DATASET **int** **XRT_DATASET1**

Specifies which dataset to use to apply a style to a point. Valid values are `XRT_DATASET1` and `XRT_DATASET2`.

XRT_POINTSTYLE_SET **int** **0**
XRT_POINTSTYLE_POINT **int** **0**

Specifies the set and point indices to use when applying a point style. `XRT_POINTSTYLE_DATASET` must be properly set prior to using these resources. Valid values for `XRT_POINTSTYLE_POINT` are `XRT_POINT_ALL` (all points in a set, including the Legend) and `XRT_POINT_LEGEND` (for setting attributes for the Legend only).

XRT_POINTSTYLE_DISPLAY **XrtDisplay (enum)** **XRT_DISPLAY_SHOW**

Used to implement missing slices in pie charts. Valid values are `XRT_DISPLAY_SHOW`, `XRT_DISPLAY_HIDE` and `XRT_DISPLAY_EXCLUDE`.

Note: If the set is not marked as `XRT_DISPLAY_SHOW`, then setting an individual point to `XRT_DISPLAY_SHOW` will have no effect.

XRT_POINTSTYLE_SLICE_OFFSET **float** **0.0**

Used to implement slice offsets. The float value is specified as a percentage of the pie radius.

XRT_POINTSTYLE_SLICE_OFFSET_USE_DEFAULT **Boolean** **True**

When True, XRT_POINTSTYLE_SLICE_OFFSET is determined by XRT/graph at render-time. When an explicit value has been set for XRT_POINTSTYLE_SLICE_OFFSET, XRT/graph sets this Boolean to False.

XRT_POINTSTYLE_DATA_STYLE **XrtDataStyle *** **dynamic**

Used to implement point-oriented data styles.

Note: Setting an **XrtDataStyle** for a given PointStyle causes *all* attributes fo the datastyle to be affected for that particular set and point.

XRT_POINTSTYLE_DATA_STYLE_USE_DEFAULT **Boolean** **True**

When True, XRT_POINTSTYLE_DATA_STYLE is determined by XRT/graph at render-time. When an explicit value has been set for XRT_POINTSTYLE_DATA_STYLE, XRT/graph sets this Boolean to False.

XRT_POINTSTYLE_PATTERN_BACKGROUND_COLOR **Char *** **NULL**

Specifies a background color for pattern filling.

XRT_POINTSTYLE_USE_DEFAULT **Boolean** **True**

Unlike other USE_DEFAULT resources, XRT_POINTSTYLE_USE_DEFAULT doesn't have a single resource counterpart, but works with ALL of the PointStyle Object resources.

When True, values for all resources of the PointStyle Object are determined by XRT/graph at render-time. When an explicit value has been set for any PointStyle object resource, XRT/graph sets this Boolean to False.

Explicitly setting XRT_POINTSTYLE_USE_DEFAULT to False has the effect of taking a snapshot fo the current PointStyle attributes and storing them in the PointStyle. Any values that were default values are explicitly set to the widget's current value for that attribute. However, there are two exceptions to this (attributes whose value XRT/graph needs to determine dynamically). They are XRT_POINTSTYLE_PATTERN_BACKGROUND_COLOR, which is set to NULL, and XRT_POINTSTYLE_DISPLAY, which is set to XRT_DISPLAY_SHOW. These values tell XRT/graph to calculate the attribute value at display time.

D

Procedures and Methods Reference

This appendix lists the XRT/graph procedures and methods in alphabetical order. Procedures marked "(source)" are also supplied as source code. Source routines are located in *$XRTHOME/src/graph/util*. If you want to change a source routine, it should be copied to another directory (and linked ahead of XRT/graph if its name is not changed).

XrtArgValToFloat()

Converts an `XtArgVal` to a float. This procedure is the opposite of *XrtFloatToArgVal()*.

```
float
XrtArgValToFloat(
    XtArgVal    value
    )
```

XrtArrCheckAxisBounds()

Determines if additional array data will require a graph repaint.

```
int
XrtArrCheckAxisBounds(
    Widget      graph,
    int         dataset,
    int         numpoints
    )
```

graph is the graph handle. *dataset* indicates whether data is being added to the graph's first or second dataset and should be either 1 or 2. *numpoints* is the number of points that have been added to each set in the dataset.

Before calling this procedure, a graph widget must have already been created and must have an Array data object attached to it using either the `XtNxrtData` or `XtNxrtData2` resource.

To use this procedure, first add *numpoints* additional data values to each set in the array. *XrtArrDataAppendPts()* is a useful convenience routine to use for this purpose. Be sure to add the points to the data object currently being pointed to by the widget's `XtNxrtData` or `XtNxrtData2` resource.

Once you've added the additional data, call *XrtArrCheckAxisBounds()*. If all of the additional data falls within the current axes bounds, this routine will return 0. Otherwise, a bit pattern is returned indicating which axis bounds were exceeded by the new data:

```
#define XRT_LTX 0x01 /* New data < X axis min */
#define XRT_GTX 0x02 /* New data > X axis max */
#define XRT_LTY 0x04 /* New data < Y axis min */
#define XRT_GTY 0x08 /* New data > Y axis max */
```

The returned value can be used in a scrolling strip chart application to determine when to scroll the "data window."

XrtArrDataAppendPts() (source)

Appends a new point to each of the sets in an Array data object.

```
int
XrtArrDataAppendPts(
    XrtDataHandle   data,
    float           xval,
    float           *yvector
    )
```

xval is the new X-value. *yvector* points to an array of Y-values (one for each set). *XrtArrDataAppendPts()* returns 1 on success and 0 on failure. For backward compatibility, *data* can also be a pointer to an `XrtData` structure.

XrtArrDataFastUpdate()

Displays additional array data in an existing graph as quickly as possible.

```
int
XrtArrDataFastUpdate(
    Widget      graph,
    int         dataset,
    int         numpoints
    )
```

graph is the graph handle. *dataset* indicates whether data is being added to the graph's first or second dataset and should be either 1 or 2. *numpoints* is the number of points that have been added to each set in the dataset.

Before calling this procedure, a graph widget must have already been created and must have an Array data object attached to it using either the `XtNxrtData` or `XtNxrtData2` resource.

To use this procedure, first add *numpoints* additional data values to each set in the array. *XrtArrDataAppendPts()* is a useful convenience routine for this purpose. Be sure to add the points to the data object currently being pointed to by the widget's `XtNxrtData` or `XtNxrtData2` resource.

Once you've added the additional data, call *XrtArrDataFastUpdate()*. This routine will graph the newly added points. If a fast data update is possible, the new points will be added extremely quickly, otherwise a graph repaint will take place (as if you had simply reset the `XtNxrtData` or `XtNxrtData2` resource). This routine returns 0 if a graph repaint was required, and 1 if a fast update was possible.

You can determine ahead of time if a fast update is possible by using the *XrtArrCheckAxisBounds()* procedure.

XrtArrDataRemovePts() (source)
Removes the point indexed by *point* from all the sets in an Array data object.

```
int
XrtArrDataRemovePts(
    XrtDataHandle   data,
    int             point
    )
```

XrtArrDataRemovePts() returns 1 on success and 0 on failure. For backward compatibility, *data* can also be a pointer to an `XrtData` structure.

XrtArrDataShiftPts() (source)
Copies a block of points in an Array data object. Points in all sets are shifted. Shifted-out points are set to the hole value.

```
int
XrtArrDataShiftPts(
    XrtDataHandle   data,
    int             dest,
    int             src,
    int             npoints
    )
```

For backward compatibility, *data* can also be a pointer to an `XrtData` structure. *src* specifies the first point in the block to be shifted; *npoints* specifies the size of the block. *dest* specifies the point to shift the block to. *src* and *dest* are zero-based.

The `npoints` attribute of the data object is not changed. This procedure returns 1 on success and 0 on failure.

XrtCustomFormatValidate()
Determines whether a given annotation format specification string is valid.

```
int
XrtCustomFormatValidate(
    char        *format
    )
```

format is a string specifying an annotation format. This procedure returns 0 on success or non-zero on failure.

XrtDataConcat() (source)
Returns a handle to a new data object created by concatenating the two existing datasets specified by *data1* and *data2*. Returns `NULL` on failure.

```
XrtDataHandle
XrtDataConcat(
    XrtDataHandle   data1,
    XrtDataHandle   data2
    )
```

For backward compatibility, either parameter can also be a pointer to an `XrtData` structure. *XrtDataConcat()* returns an `XrtData` structure only if both parameters are

XrtData structures. The type of the result is dependent on the types of *data1* and *data2*:

data1	data2	Result
NULL	anything	data2
anything	NULL	data1
General	General	General
General	Array	General
Array	General	General
Array	Array	General or Array

When concatenating an Array with an Array, an Array will be returned if both arrays have the same number of points per set, and both have the same X-values. Otherwise a General will be returned.

If either *data1* or *data2* has zero points or sets, it will be treated as if it were NULL.

XrtDataConvertToHandle()

Copies an existing XrtData structure (used in previous releases of XRT/graph) to a new data object referenced by an XrtDataHandle. Returns a NULL handle on error.

```
XrtDataHandle
XrtDataConvertToHandle(
    XrtData     *data
    )
```

This procedure does not free the existing *data*.

XrtDataCopy() (source)

Returns an exact replica of the XrtDataHandle object passed in.

```
XrtDataHandle
XrtDataCopy(
    XrtDataHandle  data
    )
```

XrtDataCopy() returns NULL on failure. For backward compatibility, *data* can also be a pointer to an XrtData structure, and if so, a pointer to an XrtData structure is returned.

XrtDataCreate()

Allocates an empty data object, using XRT_HUGE_VAL as the hole value.

```
XrtDataHandle
XrtDataCreate(
    XrtDataType data_style,
    int         nsets,
    int         npoints
    )
```

data_style must be either XRT_DATA_GENERAL or XRT_DATA_ARRAY. When the style is XRT_DATA_GENERAL, *npoints* refers to the maximum number of points in each set.

XrtDataCreateFromFile()

Allocates a data object and loads it with data read in from a text file. Returns the handle to the data object.

```
XrtDataHandle
XrtDataCreateFromFile(
    char        *filename,
    char        *errbuf
    )
```

filename is the pathname of the data file. If *XrtDataCreateFromFile()* encounters any errors, it will return NULL, and will write an error message in *errbuf*. If *errbuf* is NULL, error messages will be written to *stderr*. *errbuf* should be at least 100 bytes.

Lines in the file beginning with a "#" symbol in the first position are treated as comments and ignored. The first non-comment line must begin with either "ARRAY" or "GENERAL" followed by 2 integers specifying the number of sets (*nsets*) and the number of points (*npoints*) in each set. (If GENERAL is specified, the second integer specifies the *maximum* number of points in each set.) The two integers may optionally be followed by the character "T" to indicate that the data which follows is transposed.

The second non-comment line *can optionally* specify a hole value with the text "HOLE <value>". When XRT/graph sees this value in the dataset, it treats that point as missing. The default hole value is XRT_HUGE_VAL.

If ARRAY is specified, the next *npoints* values are treated as the X-values. The following *npoints* values are treated as the first set's Y-values. The following *npoints* values are treated as the second set's Y-values, etc.

ARRAY DATA

An example of an ARRAY specification follows:

```
# The Michelle's Microchips mm63 file
ARRAY 2 4
HOLE 10000
# X-values
1.0 2.0 3.0 4.0
# Y-values line 1 - Expenses
150.0 175.0 160.0 170.0
# Y-values line 2 - Revenue
125.0 100.0 225.0 300.0
```

ARRAY DATA with "T"

An example of the same array data in transposed form follows:

```
# The Michelle's Microchips mm63 file transposed
ARRAY 2 4 T
# X-values      Y1 Expenses     Y2 Revenues
1.0             150.0           125.0
2.0             175.0           100.0
3.0             160.0           225.0
4.0             170.0           300.0
```

If GENERAL is specified, each of the set definitions begins with an integer specifying the number of points in the set, followed by each of the set's X-values, followed by each of the set's Y-values.

GENERAL DATA

An example of a `GENERAL` specification follows:

```
# 3 lines, max 5 points each
GENERAL 3 5
HOLE 10000
# line 1, 3 points
3
1 3 5
5 14 24
# line 2, 5 points
5
1 2 3 4 5
11 15 18 21 22.
# line 3, 2 points
2
2 4
16 15.2
```

GENERAL DATA with "T"

`GENERAL` data may also be specified with the "T" transpose option.

```
# 3 lines, max 5 points each, transposed
GENERAL 3 5 T
# line 1, 3 points
3
# X     Y
1       5
3       14
5       24
# line 2, 5 points
5
# X     Y
1       11
2       15
3       18
4       21
5       22
# line 3, 2 points
2
# X     Y
2       16
4       15.2
```

XrtDataDestroy()

Destroys a data object. Use it only for objects created with *XrtDataCreate()* and *XrtDataCreateFromFile()*.

```
void
XrtDataDestroy(
    XrtDataHandle   data
    )
```

data is handle of the data object to be destroyed. If it is being used by any graph, it is automatically detached.

XrtDataExtractSet() (source)
Returns a handle to a new data object containing a copy of the set indexed by *set* of the data object referenced by *data*.

```
XrtDataHandle
XrtDataExtractSet(
    XrtDataHandle   data,
    int             set
    )
```

The returned data object is the same type (Array or General) as the object passed in. *XrtDataExtractSet()* returns `NULL` on failure. For backward compatibility, *data* can also be a pointer to an `XrtData` structure, and if so, a pointer to an `XrtData` structure is returned.

XrtDataGetDisplay()
Determines whether a given *set* is displayed, hidden or ignored.

```
XrtDisplay
XrtDataGetDisplay(
    XrtDataHandle   data,
    int             set
    )
```

This procedure returns -1 on error. Use *XrtDataSetDisplay()* to change how to display a given set.

XrtDataGetFirstPoint()
Returns the index (zero-based) of the first point to be graphed in a given *set*.

```
int
XrtDataGetFirstPoint(
    XrtDataHandle   data,
    int             set
    )
```

set is ignored for Array data objects. This procedure returns -1 on error.

XrtDataGetFirstSet()
Returns the index (zero-based) of the first set to be graphed in a given data object. Returns -1 on error.

```
int
XrtDataGetFirstSet(
    XrtDataHandle   data
    )
```

XrtDataGetHole()
Determines the hole value for a given data object. Returns `-XRT_HUGE_VAL` on error.

```
float
XrtDataGetHole(
    XrtDataHandle   data
    )
```

XrtDataGetLastPoint()

Returns the index (zero-based) of the last point to be graphed in a given *set*.

```
int
XrtDataGetLastPoint(
    XrtDataHandle   data,
    int             set
    )
```

set is ignored for Array data objects. This procedure returns -1 on error.

XrtDataGetLastSet()

Returns the index (zero-based) of the last set to be graphed in a given data object. Returns -1 on error.

```
int
XrtDataGetLastSet(
    XrtDataHandle   data,
    )
```

XrtDataGetNPoints()

Returns the total number of points in a given *set*.

```
int
XrtDataGetNPoints(
    XrtDataHandle   data,
    int             set
    )
```

set is ignored for Array data objects. This procedure returns -1 on error.

XrtDataGetNSets()

Returns the total number of sets in a given data object. This procedure returns -1 on error.

```
int
XrtDataGetNSets(
    XrtDataHandle   data
    )
```

XrtDataGetType()

Determines whether the given data object is Array or General. This procedure returns -1 on error.

```
XrtDataType
XrtDataGetType(
    XrtDataHandle   data
    )
```

XrtDataGetXData()

Gets the *internal* pointer to the given *set*'s array of X-values for updating. You can modify these X-values directly. Calling this procedure disables any other changes to the data object until you call *XrtDataRelease()*.

```
float *
XrtDataGetXData(
    XrtDataHandle   data,
    int             set
    )
```

set is ignored for Array data objects. This procedure returns `NULL` on error.

XrtDataGetXElement()

Returns the X-value of a given *set* and *point*.

```
float
XrtDataGetXElement(
    XrtDataHandle   data,
    int             set,
    int             point
    )
```

set is ignored for Array data objects. This procedure returns `-XRT_HUGE_VAL` on error.

XrtDataGetYData()

Gets the *internal* pointer to the given *set*'s array of Y-values for updating. You can modify these Y-values directly. Calling this procedure disables any other changes to the data object until you call *XrtDataRelease()*.

```
float *
XrtDataGetYData(
    XrtDataHandle   data,
    int             set
    )
```

This procedure returns `NULL` on error.

XrtDataGetYElement()

Returns the Y-value of a given *set* and *point*.

```
float
XrtDataGetYElement(
    XrtDataHandle   data,
    int             set,
    int             point
    )
```

This procedure returns `-XRT_HUGE_VAL` on error.

XrtDataRelease()

Returns control of the given data object to XRT/graph. Calls to data object methods that set attributes are re-enabled. All pointers retrieved using *XrtDataGetXData()* or *XrtDataGetYData()* should are now considered stale and should no longer be used.

```
int
XrtDataRelease(
    XrtDataHandle   data
    )
```

This procedure returns 1 on success and 0 on error.

XrtDataRemoveSet() (source)

Removes the set indexed by *set* from either an Array or General data object.

```
int
XrtDataRemoveSet(
    XrtDataHandle   data,
    int             set
    )
```

Returns 1 on success and 0 on failure. For backward compatibility, *data* can also be a pointer to an `XrtData` structure.

XrtDataSaveToFile()

Writes data stored in a data object to a text file in a format suitable for use with *XrtDataCreateFromFile()*.

```
int
XrtDataSaveToFile(
    XrtDataHandle   data,
    char            *filename,
    char            *errbuf
    )
```

data is the handle of the data object to be written. *filename* is the pathname of the file to be written. If any errors are encountered, messages will be written to *errbuf* (which should be at least 100 bytes long). If *errbuf* is `NULL`, messages will be written to *stderr*. This procedure returns 1 on success and 0 on failure. *XrtDataSaveToFile()* writes the hole value in the data object to the second line of the file.

XrtDataSetDisplay()

Changes how to display a given *set*.

```
int
XrtDataSetDisplay(
    XrtDataHandle   data,
    int             set,
    XrtDisplay      display
    )
```

display is the display status. It can be either `XRT_DISPLAY_SHOW`, `XRT_DISPLAY_HIDE` or `XRT_DISPLAY_EXCLUDE`. *XrtDataSetDisplay()* returns 1 on success and 0 on failure.

When the graph type is HiLo, HiLoOpenClose, or candle, excluding any set within a logical set excludes the entire bar. Hiding either the high or low set hides the entire bar. Hiding either the open or close set in a candle chart hides them both. For a

bubble chart, excluding/hiding either set within a logical set excludes/hides the entire logical set.

XrtDataSetFirstPoint()

Specifies the first point to be graphed in a given *set*.

```
int
XrtDataSetFirstPoint(
    XrtDataHandle   data,
    int             set,
    int             point
    )
```

set is ignored for Array data objects. *point* is the index of the first point to graph. This procedure returns 1 on success and 0 on failure.

XrtDataSetFirstSet()

Specifies the first set to be graphed in a given data object. Sets before the given set are excluded from the graph.

```
int
XrtDataSetFirstSet(
    XrtDataHandle   data,
    int             set,
    )
```

set is the index of the first set to graph. This procedure returns 1 on success and 0 on failure.

XrtDataSetHole()

Specifies the data value to use to represent a hole (missing data). Returns 1 on success and 0 on failure.

```
int
XrtDataSetHole(
    XrtDataHandle   data,
    float           hole
    )
```

XrtDataSetLastPoint()

Specifies the last point to be graphed in a given *set*.

```
int
XrtDataSetLastPoint(
    XrtDataHandle   data,
    int             set,
    int             point
    )
```

set is ignored for Array data objects. *point* is the index of the last point to graph. This procedure returns 1 on success and 0 on failure.

XrtDataSetLastSet()

Specifies the last set to be graphed in a given data object. Sets after the given set are excluded from the graph.

```
int
XrtDataSetLastSet(
    XrtDataHandle   data,
    int             set,
    )
```

set is the index of the last set to graph. This procedure returns 1 on success and 0 on failure.

XrtDataSetNPoints()

Changes the total number of points in a given *set* to *npoints*. Resets any data subset defined (using *XrtDataSetFirstPoint()* / *XrtDataSetLastPoint()*).

```
int
XrtDataSetNPoints(
    XrtDataHandle   data,
    int             set,
    int             npoints
    )
```

set is ignored for Array data objects. If *npoints* is greater than it was previously, new points are set to the hole value. This procedure returns 1 on success and 0 on failure.

XrtDataSetNSets()

Changes the total number of sets in a given data object to *nsets*. Resets any data subset defined (using *XrtDataSetFirstSet()* / *XrtDataSetLastSet()*).

```
int
XrtDataSetNSets(
    XrtDataHandle   data,
    int             nsets
    )
```

If *nsets* is greater than it was previously, new sets in Array data are filled with points set to the hole value; new sets in General data contain no points. This procedure returns 1 on success and 0 on failure.

XrtDataSetType()

Changes the data type of a given data object. Only changing Array data to General is supported. This procedure copies the Array data's common X-values to each set in the General layout.

```
int
XrtDataSetType(
    XrtDataHandle   data,
    XrtDataType     type
    )
```

type can be either `XRT_DATA_ARRAY` or `XRT_DATA_GENERAL`. This procedure returns 1 on success and 0 on failure.

XrtDataSetXData()

Sets one or more X-values in a given *set*. The X-values you specify are copied to the data object's internal X-values.

```
int
XrtDataSetXData(
    XrtDataHandle   data,
    int             set,
    float           *xvalues,
    int             n,
    int             index
    )
```

set is ignored for Array data objects. *xvalues* points to an array of X-values, which you can free after use. *n* is the number of points to change; *index* is the starting point. *XrtDataSetXData()* returns 1 on success and 0 on failure.

XrtDataSetXElement()

Sets the X-value at a particular *set* and *point* index.

```
int
XrtDataSetXElement(
    XrtDataHandle   data,
    int             set,
    int             point,
    float           x
    )
```

x is the new X-value. *set* is ignored for Array data objects. This procedure returns 1 on success and 0 on failure.

XrtDataSetYData()

Sets one or more Y-values in a given *set*. The Y-values you specify are copied to the data object's internal Y-values.

```
int
XrtDataSetYData(
    XrtDataHandle   data,
    int             set,
    float           *yvalues,
    int             n,
    int             index
    )
```

yvalues points to an array of Y-values, which you can free after use. *n* is the number of points to change; *index* is the starting point. *XrtDataSetYData()* returns 1 on success and 0 on failure.

XrtDataSetYElement()
Sets the Y-value at a particular *set* and *point* index.

```
int
XrtDataSetYElement(
    XrtDataHandle   data,
    int             set,
    int             point,
    float           y
    )
```

y is the new Y-value. This procedure returns 1 on success and 0 on failure.

XrtDataSort() (source)
Sorts the points in each set by increasing X-value.

```
int
XrtDataSort(
    XrtDataHandle   data
    )
```

XrtDataSort() returns 0 on failure and 1 on success. For backward compatibility, *data* can also be a pointer to an `XrtData` structure.

XrtDataTranspose() (source)
Transposes the sets and points of a given data object and returns a handle to a data object that contains the new data.

```
XrtDataHandle
XrtDataTranspose(
    XrtDataHandle   data
    )
```

XrtDataUpdateDataValue() (source)
Changes a data value to a new value throughout a data object. Replaces Y-values only.

```
int
XrtDataUpdateDataValue(
    XrtDataHandle   data,
    float           oldValue,
    float           newValue
    )
```

oldValue is the data value to update; *newValue* is the new value.
XrtDataUpdateDataValue() returns 0 on failure and 1 on success.

XrtDeletePointLabel()
XrtDeletePointLabel2()

Deletes one label from the graph's array of Point-labels. *XrtDeletePointLabel2()* operates on the second set of Point-labels used in combination graphs.

```
Boolean
XrtDeletePointLabel(
    Widget      graph,
    int         index,
    )
```

graph is the graph handle. The (zero-based) *index* specifies the index of the Point-label to be deleted. All Point-labels with index greater than *index* are shifted down. These procedures return `True` on success and `False` on failure.

XrtDeleteSetLabel()
XrtDeleteSetLabel2()

Deletes one label from the graph's array of Set-labels. *XrtDeleteSetLabel2()* operates on the second set of Set-labels used in combination graphs.

```
Boolean
XrtDeleteNthSetLabel(
    Widget      graph,
    int         index,
    )
```

graph is the graph handle. The (zero-based) *index* specifies the index of the Set-label to be deleted. All Set-labels with index greater than *index* are shifted down. These procedures return `True` on success and `False` on failure.

XrtDestroyData()

Destroys an `XrtData` structure. Use it only for structures created with *XrtMakeData()* and *XrtMakeDataFromFile()*.

NOTE: This procedure exists only for backward compatibility. Applications should use the *XrtDataDestroy()* procedure to destroy data objects created with *XrtDataCreate()* and *XrtDataCreateFromFile()*.

```
void
XrtDestroyData(
    XrtData     *data,
    Boolean     all
    )
```

data is the structure to be destroyed. If *all* is `True`, then any space allocated for data values is also freed.

If *data* is in use by any graph, it will be automatically detached.

XrtDrawPS()

Writes an encapsulated PostScript (EPSF-2.0) image of the graph to a file. This procedure is recommended (instead of *XrtOutputPS()*). Whereas *XrtOutputPS()* takes the pixels on the screen and outputs them using the PostScript *image* operator, *XrtDrawPS()* outputs a true PostScript file using device-independent PostScript operators. PostScript files generated by *XrtDrawPS()* are much smaller than those

created by *XrtOutputPS()* and will print at full printer resolution. *XrtDrawPS()* has several arguments that can be used to customize the PostScript.

Note that the widget must be realized (but not necessarily mapped to the display) before calling this procedure.

```
int
XrtDrawPS(
    Widget  graph,
    FILE    *fp,
    char    *msg,
    int     inches,
    double  paper_width,
    double  paper_height,
    double  margin,
    int     landscape,
    double  x_offset,
    double  y_offset,
    double  width,
    double  height,
    char    *header_font,
    int     header_size,
    char    *footer_font,
    int     footer_size,
    char    *anno_font,
    int     anno_size,
    char    *legend_font,
    int     legend_size,
    int     fill_background,
    int     smart_mono,
    int     showpage
    )
```

XrtDrawPS() returns 0 on failure and 1 on success. *graph* is the handle of the graph to be output. *fp* should point to a file which is already open and ready for writing. If *XrtDrawPS()* encounters problems, it will return a message in *msg* (*msg* should be at least 100 bytes). If *msg* is `NULL`, any error message will be written to *stderr*.

If *inches* is `True`, values provided for the paper width, height and margin, and those provided for image offset width and height will be interpreted in inches. Otherwise they will be interpreted in centimeters.

paper_width, *paper_height* and *margin* must be specified. *margin* defines the width of a boundary around the edge of the page. It is typically used for printers that can't print to the edge of a page. For a standard letter sized page destined for a LaserWriter, the following values for width, height and margin (in inches) are suggested: 8.5, 11.0 and 0.25.

If *landscape* is `True`, then the output will print sideways along the length of the page.

x_offset and *y_offset* specify the offset of the lower left corner of the image relative to the lower left corner of the paper. For example, to put the lower left corner of the image 2 inches above the bottom, and 3 inches in from the left side of the page, set *inches* to `True`, *x_offset* to 3.0 and *y_offset* to 2.0.

width and *height* specify the size of the image. Setting both to 0.0 will cause the image to print as large as possible while maintaining the original aspect ratio. If exactly one of these two parameters is non-zero, its value will be honored, and the original aspect

ratio will be maintained. If both *width* and *height* are non-zero, they will both be honored, and the aspect ratio may be adjusted (i.e. the image will be stretched to fit the specified width and height).

PostScript font names may be specified for the header, footer, legend and annotation. If `NULL` is specified, *XrtDrawPS()* will query the X Server about the X font being used on-screen, and will attempt to pick a similar font from the list of 35 Standard LaserWriter II fonts. Since not all X Servers provide the same information about fonts, this can result in different PostScript fonts when the application is run on different X Servers. It is recommended that font names be hardcoded when using *XrtDrawPS()*. For example, *header_font* can be specified as "Times-Roman". *XrtDrawPS()* ignores the font size parameters.[1]

If *fill_background* is `False`, *XrtDrawPS()* will set the background color to white. Also, when printing on a monochrome printer, the foreground color will be set to black.

If *smart_mono* is `XRT_PS_COLOR_AS_IS`, colors print in the color used on the screen; if `XRT_PS_COLOR_MONO`, colors are mapped to distinctive gray shades; if `XRT_PS_COLOR_AUTO`, the code detects[2] whether the output device is color or monochrome and prints colors or gray shades accordingly. This argument is ignored on a monochrome X server.

```
Times-Roman             Helvetica               NewCenturySchlbk-Roman
Times-Bold              Helvetica-Bold          NewCenturySchlbk-Bold
Times-Italic            Helvetica-Oblique       NewCenturySchlbk-Italic
Times-BoldItalic        Helvetica-BoldOblique   NewCenturySchlbk-BoldItalic

Palatino-Roman          Bookman-Light           Helvetica-Narrow
Palatino-Bold           Bookman-Demi            Helvetica-Narrow-Bold
Palatino-Italic         Bookman-LightItalic     Helvetica-Narrow-Oblique
Palatino-BoldItalic     Bookman-DemiItalic      Helvetica-Narrow-BoldOblique

Courier                 AvantGarde-Book         Symbol
Courier-Bold            AvantGarde-Demi         ZapfChancery-MediumItalic
Courier-Oblique         AvantGarde-BookOblique  ZapfDingbats
Courier-BoldOblique     AvantGarde-DemiOblique
```

Figure 92 The 35 Standard LaserWriter II Font Names

If *showpage* is `True`, *XrtDrawPS()* will put a PostScript *showpage* operator at the end of the EPSF file. It is possible to print several graphs on one page by making several calls to *XrtDrawPS()* using the same file pointer, and setting showpage `False` in all but the last call. When printing several graphs on a single page, position each graph using *x_offset, y_offset, width* and *height*.

1. The font size parameters (`header_size`, `footer_size`, `anno_size`, and `legend_size`) are remnants from an earlier version of XRT/graph and exist only for backward compatibility.

2. Some monochrome PostScript printers (such as the SPARCprinter) incorrectly identify themselves as color printers. To force *smart_mono* output on these printers, use `XRT_PS_COLOR_MONO`.

The following code outputs a graph (to a file) as large as possible (maintaining the aspect ratio) on a standard letter-size page with a half-inch margin. All fonts will default.

```
char msg[100];
FILE *fp;
fp = fopen("mygraph.ps", "w");
if (fp) {
    XrtDrawPS(mygraph, fp, msg,
        TRUE,                /* Use inches */
        8.5, 11.0, 0.5,      /* 8.5x11, 1/2" margin*/
        FALSE,               /* Portrait */
        0.0, 0.0, 0.0, 0.0, /* Full Size */,
        NULL, 0, NULL, 0,    /* Default Fonts */
        NULL, 0, NULL, 0,
        TRUE,                /* Keep background */
        XRT_PS_COLOR_AS_IS, /* Don't adjust */
        TRUE);               /* Show the page */
}
```

XrtDupDataStyles()

Duplicates an array of `XrtDataStyle` structures.

```
XrtDataStyle **
XrtDupDataStyles(
    XrtDataStyle    **ds
    )
```

XrtDupDataStyles() returns a pointer to the duplicate array.

XrtDupStrings()

Duplicates an array of strings.

```
char **
XrtDupStrings(
    char            **s
    )
```

XrtDupStrings() returns a pointer to the duplicate array.

XrtDupValueLabels()

This routine returns a copy of the passed Value-labels.

```
XrtValueLabel **
XrtDupValueLabels(
    XrtValueLabel   **label
    )
```

XrtEnableImageType()

Registers a new image type with XRT/graph. The image type can be predefined (such as PNG or JPEG) or user defined.

```
Boolean
XrtEnableImageType(
    XtPointer       class,
    XtPointer       functions
    )
```

class is the widget class (i.e. `xtXrtGraphWidgetClass`). *functions* is a pointer to an `XrtPDSImageFuncs` structure (see Registering Image Types on page 144 for details).

XrtFloatToArgVal()

Converts a float value to `XtArgVal`, for use when setting any of XRT/graph's floating point resources with *XtSetValues()* or *XtVaSetValues()*. Do not use with *XrtSetValues()*.

```
XtArgVal
XrtFloatToArgVal(
    float   value
    )
```

XrtFreeDataStyles()

Frees the memory used by an array of `XrtDataStyle` structures.

```
void
XrtFreeDataStyles(
    XrtDataStyle    **ds
    )
```

XrtFreePropString()

Frees the memory used by a string allocated by *XrtGetPropString()*.

```
void
XrtFreePropString(
    char            *str
    )
```

XrtFreeStrings()

Frees the memory used by an array of strings.

```
void
XrtFreeStrings(
    char            **s
    )
```

XrtFreeTextHandles()

Frees the memory used by an array of handles to text areas.

```
void
XrtFreeTextHandles(
    XrtTextHandle   *list
    )
```

XrtFreeValueLabels()

Frees the memory used by a set of Value-labels.

```
void
XrtFreeValueLabels(
    XrtValueLabel   **label
    )
```

XrtGenCheckAxisBounds()
Determines if additional general data will require a graph repaint.

```
int
XrtGenCheckAxisBounds(
    Widget      graph,
    int         dataset,
    int         set,
    int         numpoints
    )
```

graph is the graph handle. *dataset* indicates whether data is being added to the graph's first or second dataset and should be either 1 or 2. *numpoints* is the number of points that have been added to the set indexed by *set*.

Before calling this procedure, a graph widget must have already been created and must have a General type data object attached to it using either the `XtNxrtData` or `XtNxrtData2` resource.

To use this procedure, first add *numpoints* additional data values to one of the sets in the general dataset. *XrtGenDataAppendPt()* is a useful convenience routine for this purpose. Be sure to add the points to the data object currently being pointed to by the widget's `XtNxrtData` or `XtNxrtData2` resource.

Once you've added the additional data, call *XrtGenCheckAxisBounds()*. If all of the additional data falls within the current axes bounds, this routine will return 0. Otherwise, a bit pattern is returned indicating which axes bounds were exceeded by the new data:

```
#define XRT_LTX 0x01 /* New data < X axis min */
#define XRT_GTX 0x02 /* New data > X axis max */
#define XRT_LTY 0x04 /* New data < Y axis min */
#define XRT_GTY 0x08 /* New data > Y axis max */
```

The returned value can be used in a scrolling strip chart application to determine when to scroll the "data window."

XrtGenDataAppendPt() (source)
Appends the (x,y) value *xval* and *yval* to the end of the set indexed by *set*.

```
int
XrtGenDataAppendPt(
    XrtDataHandle   data,
    int             set,
    float           xval,
    float           yval
    )
```

For backward compatibility, *data* can also be a pointer to an `XrtData` structure. *XrtGenDataAppendPt()* returns 1 on success and 0 on failure. *data* must point to a General type data structure.

XrtGenDataFastUpdate()
Displays additional general data in an existing graph as quickly as possible.

```
int
XrtGenDataFastUpdate(
    Widget      graph,
    int         dataset,
    int         set,
    int         numpoints
    )
```

graph is the graph handle. *dataset* indicates whether data is being added to the graph's first or second dataset and should be either 1 or 2. *numpoints* is the number of points that have been added to the set indexed by *set*.

Before calling this procedure, a graph widget must have already been created and must have a General type data object attached to it using either the `XtNxrtData` or `XtNxrtData2` resource.

To use this procedure, first add *numpoints* additional data values to one of the sets in the general dataset. *XrtGenDataAppendPt()* is a useful convenience routine for this purpose. Be sure to add the points to the data object currently being pointed to by the widget's `XtNxrtData` or `XtNxrtData2` resource.

Once you've added the additional data, call *XrtGenDataFastUpdate()*. This routine will graph the newly added points. If a fast data update is possible, the new points will be added extremely quickly, otherwise a graph repaint will take place (as if you had simply reset the `XtNxrtData` or `XtNxrtData2` resource.) This routine returns 0 if a graph repaint was required, and 1 if a fast update was possible.

You can determine ahead of time if a fast update is possible by using the *XrtGenCheckAxisBounds()* procedure.

XrtGenDataRemovePt() (source)
Removes the value indexed by *set* and *point* from a General data object.

```
int
XrtGenDataRemovePt(
    XrtDataHandle   data,
    int             set,
    int             point
    )
```

XrtGenDataRemovePt() returns 1 on success and 0 on failure. For backward compatibility, *data* can also be a pointer to an `XrtData` structure.

XrtGenDataShiftPts() (source)
Copies a block of points in one set of a General data object. Shifted-out points are set to the hole value.

```
int
XrtGenDataShiftPts(
    XrtDataHandle   data,
    int             set,
    int             dest,
    int             src,
    int             npoints
    )
```

For backward compatibility, *data* can also be a pointer to an `XrtData` structure. *set* is the set to shift points within; *src* specifies the first point in the block to be shifted; *npoints* specifies the size of the block. *dest* specifies the point to shift the block to. *set*, *dest*, and *src*, are all zero-based.

The sets's `npoints` attribute of the data object is not changed. This procedure returns 1 on success and 0 on failure.

XrtGetImage()

Returns a pointer to the XImage of a graph. This is useful for Xlib-level programmers who need access to the graph's XImage.

```
XImage *
XrtGetImage(
    Widget  graph
    )
```

graph is the handle of the graph to be output. *XrtGetImage()* will return `NULL` on failure. The XImage will be complete, even if the graph is partially obscured on screen.

XrtGetNthDataStyle()
XrtGetNthDataStyle2()

Returns a pointer to the *n*th data style currently being used by the graph.

```
XrtDataStyle *
XrtGetNthDataStyle(
    Widget      graph,
    int         index
    )
```

graph is the graph handle.The (zero-based) *index* specifies which data style should be returned. The default data style for *index* will be returned if *index* is larger than the number of currently defined data styles. The pointer returned by this method should be considered *read-only*.

XrtGetNthDataStyle2() operates on the second set of data styles used in combination graphs.

XrtGetNthFooterString()

Returns a pointer to the *n*th footer string currently being used by the graph.

```
char *
XrtGetNthFooterString(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which footer string should be returned. `NULL` is returned if *index* is larger than the number of currently defined footer strings. The pointer returned by this method should be considered *read-only*.

XrtGetNthHeaderString()

Returns a pointer to the *n*th header string currently being used by the graph.

```
char *
XrtGetNthHeaderString(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which header string should be returned. `NULL` is returned if *index* is larger than the number of currently defined header strings. The pointer returned by this method should be considered *read-only*.

XrtGetNthPointLabel()
XrtGetNthPointLabel2()

Returns a pointer to the *n*th Point-label currently being used by the graph.

```
char *
XrtGetNthPointLabel(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which Point-label should be returned. If *index* is larger than the number of currently defined Point-labels, then `NULL` is returned. The pointer returned by this method should be considered *read-only*.

XrtGetNthPointLabel2() operates on the second set of Point-labels used in graphs where Y2 is the primary axis (i.e. `XtNxrtData` is `NULL`).

XrtGetNthSetLabel()
XrtGetNthSetLabel2()

Returns a pointer to the *n*th Set-label currently being used by the graph.

```
char *
XrtGetNthSetLabel(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which Set-label should be returned. If *index* is larger than the number of currently defined Set-labels, then `NULL` is returned. The pointer returned by this method should be considered *read-only*.

XrtGetNthSetLabel2() operates on the second set of Set-labels used in combination graphs.

XrtGetOtherSliceSets()

Determines which sets are grouped into the pie chart's *other* slice, for a given point. Returns a pointer to a list of positive integers terminated by -1.

```
int *
XrtGetOtherSliceSets(
    Widget      graph,
    int         point,
    int         dataset
    )
```

graph is the graph handle. The (zero-based) *point* specifies which point to use for the set grouping. *dataset* is either `XRT_DATASET1` or `XRT_DATASET2`. The list should be freed after use.

XrtGetPropString()

Retrieves the current value of a graph resource as a string, using the reverse converters. This method is a convenient way to bypass `XtConvertAndStore`.

```
Boolean
XrtGetPropString(
    Widget          graph,
    XrtProperty     resource,
    char            **str          /* Returned */
    )
```

graph is the graph handle. *resource* specifies any XRT/graph resource (using its cross-platform name listed on page 267); *str* is a pointer to a string returned with the string representation of the resource value. *str* must be freed after use by calling *XrtFreePropString()*.

XrtGetTextHandles()

Determines the text handles currently defined for the graph. *XrtGetTextHandles()* returns the total number attached.

```
int
XrtGetTextHandles(
    Widget          graph,
    XrtTextHandle   **list  /* Returned */
    )
```

graph is the graph handle. *list* is a `NULL`-terminated list of text handles returned. This list must be freed after use by calling *XrtFreeTextHandles()*.

XrtGetValueLabel()

Determines the Value-label closest to the label you specify for an axis.

```
XrtValueLabel *
XrtGetValueLabel(
    Widget          graph,
    XrtAxis         axis,
    XrtValueLabel   *label
    )
```

graph is the graph handle. *axis* must be either `XRT_AXIS_X`, `XRT_AXIS_Y`, or `XRT_AXIS_Y2`; the *value* element in the `XrtValueLabel` structure pointed to by *label* should be filled in before calling this procedure.

XrtGetValueLabel() returns a pointer to an existing `XrtValueLabel` structure or `NULL` if any parameter is invalid. The structure's *value* and *string* elements are the closest Value-label (to the one passed in). If no Value-labels exist for an axis, the *string* element is `NULL`. The returned pointer should be considered *read-only*.

XrtGetValues()

Retrieves the current value of one or more graph resources.

```
void
XrtGetValues(
    Widget      graph,
    ...,
    NULL
    )
```

graph is the graph handle. ... is one or more *resource-value* pairs. *resource* is any XRT/graph resource (specified as its `XrtProperty` cross-platform name listed on page 267); *value* is a pointer to a variable to contain the returned resource value. The data type of *value* must be the same as the resource's data type. Do not use *XrtArgValToFloat()* to access resources of type `float`. The list is terminated by `NULL`.

XrtInsertNthDataStyle()

Inserts a new data style which becomes the *n*th data style.

```
int
XrtInsertNthDataStyle(
    Widget          graph,
    int             index,
    XrtDataStyle    *ds
    int             dataset
    )
```

graph is the graph handle. The (zero-based) *index* specifies the index of the new data style. All data styles with index greater than or equal to *index* are shifted up. *ds* points to an `XrtDataStyle` structure containing the new data style to be used. XRT/graph makes its own copy of the data style pointed to by *ds*. *dataset* is either 1 or 2, indicating that either the first or second set of data styles is to be operated on.

A side effect of this method is that the appropriate `XtNxrtDataStylesUseDefault` or `XtNxrtDataStyles2UseDefault` resource is set to `False`.

XrtInsertNthPointLabel()
XrtInsertNthPointLabel2()

Inserts a new Point-label which becomes the *n*th Point-label. *XrtInsertNthPointLabel2()* operates on the second set of Point-labels used in combination graphs.

```
Boolean
XrtInsertNthPointLabel(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies the index of the new Point-label. All Point-labels with index greater than or equal to *index* are shifted up. *s* is the

string to be used. XRT/graph makes its own copy of the characters pointed to by *s*. These procedures return `True` on success and `False` on failure.

XrtInsertNthSetLabel()
XrtInsertNthSetLabel2()

Inserts a new Set-label which becomes the *n*th Set-label. *XrtInsertNthSetLabel2()* operates on the second set of Set-labels used in combination graphs.

```
Boolean
XrtInsertNthSetLabel(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies the index of the new Set-label. All Set-labels with index greater than or equal to *index* are shifted up. *s* is the string to be used. XRT/graph copies the characters pointed to by *s*. These procedures return `True` on success and `False` on failure.

XrtMakeData()

Allocates an empty `XrtData` structure, using `XRT_HUGE_VAL` as the hole value.

NOTE: This procedure exists only for backward compatibility. Applications should use the *XrtDataCreate()* procedure to allocate data objects.

```
XrtData *
XrtMakeData(
    XrtDataType data_type,
    int         nsets,
    int         npoints,
    Boolean     all
    )
```

data_type must be either `XRT_DATA_GENERAL` or `XRT_DATA_ARRAY`. If *all* is `True`, then space for the data values is also allocated, otherwise only the structure and pointers are allocated. When the style is `XRT_DATA_GENERAL`, *npoints* refers to the maximum number of points in each set.

XrtMakeDataFromFile()

Allocates an `XrtData` structure and loads it with data read in from a text file.

NOTE: This procedure exists only for backward compatibility. Applications should use the *XrtDataCreateFromFile()* procedure to read data files.

```
XrtData *
XrtMakeDataFromFile(
    char        *filename,
    char        *errbuf
    )
```

filename is the pathname of the data file. If *XrtMakeDataFromFile()* encounters any errors, it will return `NULL`, and will write an error message in *errbuf*. If *errbuf* is `NULL`, error messages will be written to *stderr*. *errbuf* should be at least 100 bytes.

For details on the file format understood by this procedure, see the *XrtDataCreateFromFile()* procedure on page 215.

XrtMakeTime()

Converts time information into a long integer (representing the number of seconds since January 1, 1970).

```
long
XrtMakeTime(
    int             yr,
    int             mon,
    int             day,
    int             hr,
    int             min,
    int             sec,
    )
```

yr is the year minus 1900. *mon* is a value from 0 to 11 representing the month. *day* is the day of the month. *hr* is a value from 0 to 23 representing the hour of the day. *min* and *sec* are the minute and second.

XrtMap()

Maps a pixel coordinate to a graph coordinate.

```
XrtRegion
XrtMap(
    Widget           graph,
    int              yaxis,
    int              pix_x,
    int              pix_y,
    XrtMapResult    *map
    )
```

Used by an application to determine the graph coordinates corresponding to *pix_x* and *pix_y* (in the graph whose handle is specified by *graph*). Set *yaxis* to 1 for the coordinate system defined by the first Y-axis, or set to 2 for the second Y-axis. Results are returned in *map*. Typically used in an event handling procedure.

XrtMrmInitialize()

Registers the XRT/graph widget class with Mrm. For use by UIL applications. This routine must be called after *MrmInitialize()*.

```
void
XrtMrmInitialize(void)
```

XrtOutputPS()

Outputs an encapsulated PostScript (EPSF) representation of the graph to a file. The PostScript file is suitable for sending to a PostScript printer, or importing into other PostScript files. The file conforms to EPSF-2.0 format.

```
int
XrtOutputPS(
    Widget      graph,
    FILE        *fp,
    Boolean     color,
    char        *msg
    )
```

graph is the handle of the graph to be output. *fp* should point to a file which is already opened and ready for writing. If color is `True`, a PostScript *colorimage* will be created, otherwise a monochrome PostScript *image* will be created. If *XrtOutputPS()* encounters problems, it will return a message in *msg* (*msg* should be at least 100 bytes). If *msg* is `NULL`, any error message will be written to *stderr*. *XrtOutputPS()* returns 0 on failure and 1 on success. The PostScript file will be complete, even if the graph is partially obscured on screen. Color graphs can result in large PostScript files, thus care should be taken to ensure that there is enough space available for the file before calling this routine.

XrtOutputRaster()

Outputs a Sun rasterfile representation of the graph to a file. It uses the Sun *rasterfile(5)* routine, so it is only available on SPARC architectures. The rasterfile is in standard format, and all colormap information is included.

```
int
XrtOutputRaster(
    Widget      graph,
    FILE        *fp,
    char        *msg
    )
```

graph is the handle of the graph to be output. *fp* should point to a file which is already opened, and ready for writing. If *XrtOutputRaster()* encounters problems, it will return a message in *msg* (*msg* should be at least 100 bytes). If *msg* is `NULL`, any error message will be written to *stderr*. *XrtOutputRaster()* returns 0 on failure and 1 on success. The rasterfile will be complete, even if the graph is partially obscured on screen. Color graphs can result in large rasterfiles thus care should be taken to ensure that there is enough space available for the file before calling this routine.

XrtOutputXwd()

Outputs an X Window Dump representation of the graph to a file.

```
int
XrtOutputXwd(
    Widget      graph,
    FILE        *fp,
    char        *msg
    )
```

graph is the handle of the graph to be output. *fp* should point to a file which is already opened and ready for writing. If *XrtOutputXwd()* encounters problems, it will return a

message in *msg* (*msg* should be at least 100 bytes). If *msg* is NULL, any error message will be written to *stderr*. *XrtOutputXwd()* returns 0 on failure and 1 on success. The xwd file will be complete, even if the graph is partially obscured on screen.

XrtPick()

Picks the displayed data that is closest to the given pixel coordinate.

```
XrtRegion
XrtPick(
    Widget          graph,
    XrtDsGroup      ds_group,
    int             pix_x,
    int             pix_y,
    XrtPickResult   *pick,
    XrtFocus        focus
    )
```

Used by an application to determine what data (in terms of *set* and *point*) is displayed closest to the pixel at (*pix_x*, *pix_y*) in the graph whose handle is specified by *graph*. Results are returned in *pick*. Typically used in an event handling procedure.

The *ds_group* parameter is used to tell *XrtPick()* which dataset to use in combination graphs. When calling *XrtPick()* on a single graph, set this to XRT_DATASET1. On combination graphs, set it to XRT_DATASET1 or XRT_DATASET2 to restrict pick results to one of the datasets. Setting it to XRT_DATASET1 | XRT_DATASET2 causes *XrtPick()* to consider both datasets.

The *focus* parameter indicates the method used by the *XrtPick()* routine to determine the closest set and point. Setting focus to XRT_XFOCUS means that only the distance along the X-axis is used to determine "closeness". Setting it to XRT_YFOCUS means that only the distance along the Y-axis is used. Setting it to XRT_XFOCUS | XRT_YFOCUS means that both the X- and Y-axis distance is used.

XrtPointStyleCreate()

Creates a PointStyle object as a child of a graph. The PointStyle is created but not attached to a specific data point. This returns an XrtPointStyleHandle which is used in all of the other XrtPointStyle... methods.

```
XrtPointStyleHandle
XrtPointStyleCreate(
    Widget      graph
    )
```

graph is the graph handle.

The first parameter to all PointStyle API functions, except *XrtPointStyleCreate()*, is the XrtPointStyleHandle, returned by *XrtPointStyleCreate()*.

XrtPointStyleDestroy()

Destroys a PointStyle.

```
void
XrtPointStyleDestroy(
    XrtPointStyleHandle     pointstyle
    )
```

pointstyle is the PointStyle object handle.

XrtPointStyleGetValues()

Retrieves the current value of one or more PointStyle object attributes.

```
void
XrtPointStyleGetValues(
    XrtPointStyleHandle pointstyle,
    ...
    NULL
    )
```

pointstyle is the PointStyle object handle. ... is one or more *attribute-value* pairs. *attribute* is any PointStyle object attribute, as listed in the PointStyle Object Attribute Reference on page 209. *value* is a pointer to a variable to contain the returned attribute value. The data type of *value* must be the same as the attribute's data type. The list is terminated by **NULL**.

XrtPointStyleHandle()

Finds a PointStyle for a given dataset based on its exact set and point attributes. Returns the most recent PointStyle created with these attributes. Returns `NULL` if no match is found.

```
XrtPointStyleHandle
XrtPointStyleFindExact(
    Widget          graph,
    int             dataset,
    int             set,
    int             point
    )
```

graph is the graph handle. *dataset* is either `XRT_DATASET1` or `XRT_DATASET2`. *set* is the set attribute of the PointStyle to be found. *point* is the point attribute of the PointStyle to be found.

XrtPointStyleSetValues()

Sets the current value of one or more PointStyle object attributes.

```
void
XrtPointStyleSetValues(
    XrtPointStyleHandle pointstyle,
    ...
    NULL
    )
```

pointstyle is the PointStyle object handle. ... is one or more *attribute-value* pairs. *attribute* is any PointStyle object attribute, as listed in the PointStyle Object Attribute Reference on page 209. *value* specifies the new value. The data type of *value* must be the same as the attribute's data type. The list is terminated by **NULL**.

XrtPropertyEditorEnable()

Initializes the XRT/graph Property Editor, and specifies capabilities enabled for a specific graph.

```
Boolean
XrtPropertyEditorEnable(
    Widget      graph,
    Boolean     enable,
    char        *title_string,
    Boolean     allow_load_save,
    Boolean     do_updates
    )
```

graph is the graph handle. *enable* specifies whether to enable or disable the Property Editor. *title_string* is the text to display in the title bar ("XRT/graph Property Editor" by default). *allow_load_save* specifies whether the end-user is able to load/save *.XRT* and data files and generate C code. *do_updates* specifies whether to update the Property Editor whenever the graph updates. *XrtPropertyEditorEnable()* returns `True` on success, `False` on failure.

XrtRemoveNthDataStyle()

Removes the *n*th data style.

```
int
XrtRemoveNthDataStyle(
    Widget          graph,
    int             index,
    int             dataset
    )
```

graph is the graph handle. The (zero-based) *index* specifies the index of the data style to be removed. All data styles with index greater than *index* are shifted down. *dataset* is either 1 or 2, indicating that either the first or second set of data styles is to be operated on. *XrtRemoveNthDataStyle()* returns 1 on success, 0 on failure.

XrtSampleColorFunc()

This function provides an example of a function that can be used for the `XtNxrtColorFunc` resource. It is an adaptation of the XPM closest color matching algorithm (adapted to fit the semantics that `XtNxrtColorFunc` needs). The source code for this function is provided in *$XRTHOME\src\graph\util\x_color_func.c*.

```
unsigned long
XrtSampleColorFunction(
    Display     *display,
    int         screen,
    Colormap    colormap,
    XColor      *color,
    Boolean     do_free
    )
```

XrtSaveDataToFile()

Writes data stored in an `XrtData` structure to a text file in a format suitable for use with *XrtMakeDataFromFile()*.

NOTE: This procedure exists only for backward compatibility. Applications should use the *XrtDataSaveToFile()* procedure to save data files.

```
int
XrtSaveDataToFile(
    XrtData         *data,
    char            *filename,
    char            *errbuf
    )
```

data is the structure containing the data to be written. *filename* is the pathname of the file to be written. If any errors are encountered, messages will be written to *errbuf* (which should be at least 100 bytes long). If *errbuf* is `NULL`, messages will be written to *stderr*. This procedure returns 1 on success and 0 on failure. *XrtSaveDataToFile()* writes the hole value in the `XrtData` structure to the second line of the file.

XrtSaveImageAsJpeg()

Outputs a JPEG representation of the graph to a file. The JPEG image type must first be registered with XRT/graph using *XrtEnableImageType()*.

```
Boolean
XrtSaveImageAsJpeg(
    Widget          graph,
    String          filename,
    String          msg
    )
```

graph is the handle of the graph to be output. *filename* is the name of the file to which the image is to be written. If *XrtSaveImageAsJpeg()* encounters problems, it will return a message in *msg* (msg should be at least 100 bytes). If msg is `NULL`, any error message will be written to *stderr*. *XrtSaveImageAsJpeg()* returns `True` on success.

XrtSaveImageAsPNG()

Outputs a PNG representation of the graph to a file. The PNG image type must first be registered with XRT/graph using *XrtEnableImageType()*.

```
Boolean
XrtSaveImageAsPng(
    Widget          graph,
    String          filename,
    String          msg
    )
```

graph is the handle of the graph to be output. *filename* is the name of the file to which the image is to be written. If *XrtSaveImageAsPng()* encounters problems, it will return a message in *msg* (msg should be at least 100 bytes). If msg is `NULL`, any error message will be written to *stderr*. *XrtSaveImageAsPng()* returns `True` on success.

XrtSaveImageGeneric()

Outputs a representation of the graph to a file based on the image type. The image type must be registered with XRT/graph using *XrtEnableImageType()* unless it is a preregistered type (XPM or XWD).

```
Boolean
XrtSaveImageGeneric(
    Widget          graph,
    int             image_type,
    String          filename,
    String          msg
    )
```

graph is the handle of the graph 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 *XrtSaveImageGeneric()* encounters problems, it will return a message in *msg* (msg should be at least 100 bytes). If msg is `NULL`, any error message is written to *stderr*. *XrtSaveImageGeneric()* returns `True` on success.

XrtSetNthDataStyle()
XrtSetNthDataStyle2()

Sets the *n*th data style. *XrtSetNthDataStyle2()* operates on the second set of data styles used in combination graphs.

```
void
XrtSetNthDataStyle(
    Widget       graph,
    int          index,
    XrtDataStyle *ds
    )
```

graph is the handle. The (zero-based) *index* specifies which data style should be changed. *ds* points to an `XrtDataStyle` structure containing the new data style to be used. XRT/graph makes its own copy of the data style pointed to by *ds*.

A side effect of this method is that the appropriate `XtNxrtDataStylesUseDefault` or `XtNxrtDataStyles2UseDefault` resource is set to `False`.

XrtSetNthFooterString()

Sets the *n*th footer string.

```
void
XrtSetNthFooterString(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies which footer string should be changed. *s* is the new string to be used. XRT/graph makes its own copy of the characters pointed to by *s*.

XrtSetNthHeaderString()

Sets the *n*th header string.

```
void
XrtSetNthHeaderString(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies which header string should be changed. *s* is the new string to be used. XRT/graph makes its own copy of the characters pointed to by *s*.

XrtSetNthPointLabel()
XrtSetNthPointLabel2()

Sets the *n*th Point-label. Point-labels only display when `XtNxrtXAnnotationMethod` is `XRT_ANNO_POINT_LABELS`. *XrtSetNthPointLabel2()* operates on the second set of Point-labels used in combination graphs.

```
void
XrtSetNthPointLabel(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies which Point-label should be changed. *s* is the new string to be used. XRT/graph makes its own copy of the characters pointed to by *s*.

XrtSetNthSetLabel()
XrtSetNthSetLabel2()

Sets the *n*th Set-label. *XrtSetNthSetLabel2()* operates on the second set of Set-labels used in combination graphs.

```
void
XrtSetNthSetLabel(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies which Set-label should be changed. *s* is the new string to be used. XRT/graph copies the characters pointed to by *s*.

XrtSetPropString()

Sets a graph resource to the value represented by a string.

```
Boolean
XrtSetPropString(
    Widget          graph,
    XrtProperty     resource,
    char            *str
    )
```

graph is the graph handle. *resource* specifies any XRT/graph resource (using its cross-platform name listed on page 267); *str* is a pointer to a string representation to set the resource to (syntax is the same as for a resource file, see Appendix H on page 277).

XrtSetReverseConverters()

Registers XRT/graph's reverse converters for use with *XtConvertAndStore()*. Reverse converters allow applications to get resource values in a human-readable form.

```
void
XrtSetReverseConverters(void)
```

XrtSetValueLabel()

Defines a new Value-label, on either the X-, Y-, or Y2-axis.

```
void
XrtSetValueLabel(
    Widget           graph,
    XrtAxis          axis,
    XrtValueLabel   *label
    )
```

graph is the graph handle. *axis* must be either `XRT_AXIS_X`, `XRT_AXIS_Y`, or `XRT_AXIS_Y2`; the *value* and *string* elements in the `XrtValueLabel` structure pointed to by *label* should be filled in before calling this procedure. This procedure adds the new Value-label to the graph. XRT/graph copies the `XrtValueLabel` pointed to by label.

XrtSetValues()

Sets one or more graph resources.

```
void
XrtSetValues(
    Widget          graph,
    ...,
    NULL
    )
```

graph is the graph handle. ... is one or more *resource-value* pairs. *resource* is any XRT/graph resource (specified as its `XrtProperty` cross-platform name listed on page 267); *value* specifies the new value. The data type of *value* must be the same as the resource's data type. Do not use *XrtFloatToArgVal()* to set resources of type `float` with this method. The list is terminated by `NULL`.

XrtTextAreaCreate()

Creates a text object with its attributes set to default values. This procedure returns the handle to the created text object.

```
XrtTextHandle
XrtTextAreaCreate(
    Widget      parent
    )
```

XrtTextAreaDestroy()

Destroys a text object. Use it only for objects created with *XrtTextAreaCreate()*.

```
void
XrtTextAreaDestroy(
    XrtTextHandle   hText
    )
```

hText is the handle of the text object to be destroyed. If it is being used by any graph, it is automatically detached.

XrtTextAreaGetPropString()

Retrieves the current value of a text object attribute as a string, using the reverse converters.

```
Boolean
XrtTextAreaGetPropString(
    XrtTextHandle   hText,
    XrtProperty     resource,
    char            **str          /* Returned */
    )
```

hText is the handle of the text object. *resource* specifies any TextArea attribute; *str* is a pointer to a string returned with the string representation of the resource value. *str* must be freed after use by calling *XrtFreePropString()*.

XrtTextAreaGetValues()

Retrieves the current value of one or more text object attributes.

```
void
XrtTextAreaGetValues(
    XrtTextHandle   hText
    ...,
    NULL
    )
```

hText is the text object handle. ... is one or more *attribute-value* pairs. *attribute* is any text object attribute (listed in Appendix B on page 205); *value* is a pointer to a variable to contain the returned attribute value. The data type of *value* must be the same as the attribute's data type. The list is terminated by `NULL`.

XrtTextAreaSetPropString()

Sets a text object attribute to the value represented by a string.

```
Boolean
XrtTextAreaSetPropString(
    XrtTextHandle   hText,
    XrtProperty     resource,
    char            *str
    )
```

hText is the text object handle. *resource* specifies any TextArea object (using its cross-platform name listed on page 267); *str* is a pointer to a string representation to set the resource to (syntax is the same as for a resource file, see Appendix H on page 277).

XrtTextAreaSetValues()

Sets one or more text object attributes.

```
void
XrtTextAreaSetValues(
    XrtTextHandle   hText
    ...,
    NULL
    )
```

hText is the text object handle. ... is one or more *attribute-value* pairs. *attribute* is any text object attribute (listed in Appendix B on page 205); *value* specifies the new value. The data type of *value* must be the same as the attribute's data type. Do not use *XrtFloatToArgVal()* to set attributes of type `float`. The list is terminated by `NULL`.

XrtTextAttach()

Attaches a text area to a graph. **NOTE:** This procedure exists only for backward compatibility. Applications should use the text object API described in section 7.3 on page 130.

```
void
XrtTextAttach(
    Widget          graph,
    XrtTextHandle   handle
    )
```

XrtTextCreate()

Used to create a new text area. **NOTE:** This procedure exists only for backward compatibility. Applications should use the text object API described in section 7.3 on page 130.

```
XrtTextHandle
XrtTextCreate(
    Widget          graph,
    XrtTextDesc     *textd
    )
```

XrtTextDestroy()

Destroys a text area. **NOTE:** This procedure exists only for backward compatibility. Applications should use the text object API described in section 7.3 on page 130.

```
void
XrtTextDestroy(
    Widget          graph,
    XrtTextHandle   handle
    )
```

XrtTextDetach()

Detaches a text area from a graph. **NOTE:** This procedure exists only for backward compatibility. Applications should use the text object API described in section 7.3 on page 130.

```
void
XrtTextDetach(
    Widget          graph,
    XrtTextHandle   handle
    )
```

XrtTextDetail()

Retrieves text area information. Returns 1 if *handle* is valid, and 0 otherwise. **NOTE:** This procedure exists only for backward compatibility. Applications should use the text object API described in section 7.3 on page 130.

```
int
XrtTextDetail(
    Widget          graph,
    XrtTextHandle   handle,
    XrtTextDesc     *textd
    )
```

XrtTextUpdate()

Updates text area information. **NOTE:** This procedure exists only for backward compatibility. Applications should use the text object API described in section 7.3 on page 130.

```
void
XrtTextUpdate(
    Widget          graph,
    XrtTextHandle   handle,
    XrtTextDesc     *textd
    )
```

XrtUnmap()

Unmaps from a graph coordinate to a pixel coordinate.

```
void
XrtUnmap(
    Widget          graph,
    int             yaxis,
    double          x,
    double          y,
    XrtMapResult    *map
    )
```

Used by an application to determine the pixel coordinates corresponding to graph coordinate *(x, y)* in the graph whose handle is specified by *graph*. Set *yaxis* to 1 to unmap from the coordinate system defined by the first Y-axis, or set to 2 for the second Y-axis. Results are returned in the *pix_x* and *pix_y* elements of the structure pointed to by *map*. *pix_x* and *pix_y* values of -1 are returned for unmap requests that are out of range.

XrtUnpick()

Determines a pixel coordinate given a dataset *set* and *point*.

```
void
XrtUnpick(
    Widget          graph,
    int             dataset,
    int             set,
    int             point,
    XrtPickResult   *pick
    )
```

Used by an application to determine the pixel coordinates at which a data element is displayed (in the graph whose handle is specified by *graph*).

Set *dataset* to `XRT_DATASET1` unless unpicking from the second dataset of a combination graph. *set* and *point* are indices into the dataset.

Results are returned in the *pix_x* and *pix_y* elements of the structure pointed to by *pick*. Both *pix_x* and *pix_y* will be -1 if the data is out of range.

XtCreateXrtGraph()

Creates an XRT/graph widget.

```
Widget
XtCreateXrtGraph(
    Widget  parent,
    char    *name,
    Arg     *arglist,
    int     argcount
    )
```

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

XtIsXrtGraph()

Returns `True` if a widget is a subclass of an XRT/graph widget.

```
Boolean
XtIsXrtGraph(
    Widget  graph
    )
```

E

Translations and Actions Reference

Default Translations
Action Routines

E.1 Default Translations

XRT/graph inherits translations from XmPrimitive, and defines the following translations of its own:

Event	Action
Shift <Btn2Down>	ModifyStart(Translate)
Ctrl <Btn2Down>	ModifyStart(Scale)
~Shift ~Ctrl <Btn2Down>	ModifyStart(Rotate)
Ctrl <Btn1Down>	ModifyStart(Zoom)
Shift <Btn1Down>	ModifyStart(ZoomAxis)
<Key> r	ModifyStart(Reset) ViewReset() ModifyEnd(Reset)
~Shift ~Ctrl <Btn3Down>	Properties()
Shift <Btn2Motion>	TranslateUpdate()
Ctrl <Btn2Motion>	ScaleUpdate()
~Shift ~Ctrl <Btn2Motion>	RotateUpdate()
Ctrl <Btn1Motion>	ZoomUpdate()
Shift <Btn1Motion>	ZoomAxisUpdate()

Event	Action
Shift <Btn2Up>	ModifyEnd(Translate)
Ctrl <Btn2Up>	ModifyEnd(Scale)
~Shift ~Ctrl <Btn2Up>	ModifyEnd(Rotate)
Ctrl <Btn1Up>	ZoomEnd() ModifyEnd(Zoom)
Shift <Btn1Up>	ZoomAxisEnd() ModifyEnd(ZoomAxis)
Ctrl Button1 <Key> osfCancel[a]	ZoomCancel() ModifyEnd(Zoom)
Shift Button1 <Key> osfCancel[a]	ZoomAxisCancel() ModifyEnd(ZoomAxis)
Ctrl <KeyUp>	ZoomCancel()
Shift <KeyUp>	ZoomCancel()

a. When the XRT/graph widget is instantiated inside a Form or BulletinBoard widget, the widget does not receive the osfCancel event. You should define another event for the cancel action.

E.2 Action Routines

XRT/graph defines the following action routines, listed in alphabetical order:

Map()

Determines the graph coordinates that map to the pixel coordinates of a user event. By default, this action routine is not bound to any event. *Map()* calls the *XrtMap()* procedure and triggers the callbacks specified by `XtNxrtMapCallback`, passing each callback a pointer to an `XrtMapCallbackStruct` structure that contains the graph coordinates. *Map()* takes one optional parameter that specifies the Y-axis to map coordinates to:

`YAxis1`	Use the Y-axis coordinates defined by the first Y-axis
`YAxis2`	Use the Y-axis coordinates defined by the second Y-axis

If no parameter is specified, *Map()* uses the first Y-axis.

ModifyStart()

Starts any user interaction, giving the application a way to disallow it. *ModifyStart()* triggers the callbacks specified by `XtNxrtModifyCallback`, passing each callback a pointer to an `XrtModifyCallbackStruct` structure with *reason* set to `XRT_REASON_MODIFY_START`. The application can set *doit* to `False` to disable all user interaction.

ModifyStart() takes one parameter that indicates the user interaction attempted:

`Translate`	User wants to translate the graph
`Scale`	User wants to scale the graph
`Rotate`	User wants to rotate the graph
`Zoom`	User wants to zoom the graph
`ZoomAxis`	User wants to zoom the graph with axes kept visible
`Reset`	User wants to reset the graph

ModifyEnd()

Notifies the application that user interaction has ended. *ModifyEnd()* triggers the callbacks specified by `XtNxrtModifyCallback`, passing each callback a pointer to an `XrtModifyCallbackStruct` structure with *reason* set to `XRT_REASON_MODIFY_END`. *ModifyEnd()* takes one parameter that indicates the user interaction completed:

`Translate`	Interactive translation has finished
`Scale`	Interactive scaling has finished
`Rotate`	Interactive rotation has finished
`Zoom`	Interactive zooming has finished
`ZoomAxis`	Interactive axis zooming has finished
`Reset`	Reset has finished

NoAction()

Does nothing, triggering no callbacks. By default, this action routine is not bound to any event. Setting a translation to call this routine effectively disables the translation. For example, to turn off scaling, set a new translation for `Ctrl <Btn2Motion>` that calls *NoAction()*.

Pick()

Determines the graph data values closest to the pixel coordinates of a user event. By default, this action routine is not bound to any event. *Pick()* calls the *XrtPick()* procedure and triggers the callbacks specified by `XtNxrtPickCallback`, passing each callback a pointer to an `XrtPickCallbackStruct` structure that contains the graph

data values. *Pick()* takes two optional parameters (`Dataset, AxisFocus`) that specify the dataset and axis focus to use:

`Dataset`	`Data1` uses the first dataset; `Data2` uses the second dataset; `AllData` uses both datasets.
`AxisFocus`	`XFocus` uses the X-axis to determine the closest data point; `YFocus` uses the Y-axis; `XYFocus` uses both axes.

If no parameters are specified, *Pick()* uses (`AllData, XYFocus`).

Properties()

Runs the XRT/graph Property Editor to enable a user to edit resource values on a running graph. The Property Editor is displayed only when it has been enabled using *XrtPropertyPageEnable()*. No callbacks are triggered.

RotateUpdate()

Rotates bar and pie charts (if `XtNxrtGraphDepth` is > 0). *RotateUpdate()* calculates new values for the `XtNxrtGraphInclination` and `XtNxrtGraphRotation` resources, and triggers any callbacks specified by `XtNxrtRotateCallback`, passing each callback a pointer to an `XrtRotateCallbackStruct` structure. Unless the application sets `XrtRotateCallbackStruct`'s *doit* parameter to `False`, *RotateUpdate()* applies the new graph rotation values to the widget (triggering an `XtNxrtExposeCallback`).

ScaleUpdate()

Scales the graph. *ScaleUpdate()* calculates new values for the graph margins, and triggers any callbacks specified by `XtNxrtTransformCallback`, passing each callback a pointer to an `XrtTransformCallbackStruct` structure. Unless the application sets `XrtTransformCallbackStruct`'s *doit* parameter to `False`, *ScaleUpdate()* applies the new graph margins to the widget (triggering an `XtNxrtExposeCallback`).

TranslateUpdate()

Translates the graph. *TranslateUpdate()* calculates new values for the graph margins, and triggers any callbacks specified by `XtNxrtTransformCallback`, passing each callback a pointer to an `XrtTransformCallbackStruct` structure. Unless the application sets `XrtTransformCallbackStruct`'s *doit* parameter to `False`, *TranslateUpdate()* applies the new graph margins to the widget (triggering an `XtNxrtExposeCallback`).

ViewReset()

Restores all scaling, translation, and zooming to XRT/graph defaults. Only performed when there has been user-interaction with the widget. *ViewReset()* triggers any callbacks specified by `XtNxrtTransformCallback`, passing each a pointer to an `XrtTransformCallbackStruct` structure containing the default graph margin values and the *reset* parameter set to `True`. *ViewReset()* then triggers any callbacks specified by `XtNxrtZoomAxisCallback`, passing each a pointer to an `XrtZoomAxisCallbackStruct` structure containing the default axis bounds and the *reset* parameter set to `True`. Unless the application sets the structures' *doit* parameter

to `False`, *ViewReset()* applies the new graph margins and default axes minimums and maximums to the widget (triggering an `XtNxrtExposeCallback`).

ZoomAxisUpdate()
ZoomUpdate()
Updates the zoom area selection "rubberband" for axis or graphics zoom. No callbacks are triggered.

ZoomAxisEnd()
Zooms into the graph at the area selected by the "zoom rubberband", with axes kept visible. *ZoomAxisEnd()* calculates new values for axis minimums and maximums, and triggers any callbacks specified by `XtNxrtZoomAxisCallback`, passing each callback a pointer to an `XrtZoomAxisCallbackStruct` structure. Unless the application sets `XrtZoomAxisCallbackStruct`'s *doit* parameter to `False`, *ZoomAxisEnd()* applies the new axis bounds to the widget (triggering an `XtNxrtExposeCallback`).

ZoomEnd()
Zooms into the graph at the area selected by the "zoom rubberband." *ZoomEnd()* calculates new values for the graph margins, and triggers any callbacks specified by `XtNxrtTransformCallback`, passing each callback a pointer to an `XrtTransformCallbackStruct` structure. Unless the application sets `XrtTransformCallbackStruct`'s *doit* parameter to `False`, *ZoomEnd()* applies the new graph margins to the widget (triggering an `XtNxrtExposeCallback`).

ZoomAxisCancel()
ZoomCancel()
Cancels an axis or graphics zoom in progress, erasing the "zoom rubberband."

F

Data Types

This appendix lists the XRT/graph data types in alphabetical order. The C language definition of structures is also provided.

XrtAdjust

Enumeration used by the `XtNxrtFooterAdjust` and `XtNxrtHeaderAdjust` resources:

```
XRT_ADJUST_LEFT
XRT_ADJUST_RIGHT
XRT_ADJUST_CENTER
```

XrtAlign

Enumeration used with the `XtNxrtLegendOrientation` resource:

```
XRT_ALIGN_VERTICAL
XRT_ALIGN_HORIZONTAL
```

XrtAnchor

Enumeration used with the `XtNxrtLegendAnchor` resource and for text areas:

```
XRT_ANCHOR_NORTH
XRT_ANCHOR_SOUTH
XRT_ANCHOR_EAST
XRT_ANCHOR_WEST
XRT_ANCHOR_NORTHWEST
XRT_ANCHOR_NORTHEAST
XRT_ANCHOR_SOUTHWEST
XRT_ANCHOR_SOUTHEAST
XRT_ANCHOR_HOME
XRT_ANCHOR_BEST
```

`XRT_ANCHOR_HOME` and `XRT_ANCHOR_BEST` are for use in text areas only.

XrtAngleUnit

Enumeration used with the `XtNxrtAngleUnit` resource:

```
XRT_ANGLE_DEGREES
XRT_ANGLE_RADIANS
XRT_ANGLE_GRADS
```

XrtAnnoMethod

Enumeration used to define methods for X-, Y-, and Y2-axis annotation. Used with the `XtNxrt[XYY2]AnnotationMethod` resource:

```
XRT_ANNO_VALUES
XRT_ANNO_POINT_LABELS
XRT_ANNO_VALUE_LABELS
XRT_ANNO_TIME_LABELS
```

XrtAnnoPlacement

Enumeration used with the `XtNxrt[XY]AnnoPlacement` resource to select the location of axis annotation:

```
XRT_ANNO_AUTO
XRT_ANNO_ORIGIN
XRT_ANNO_MIN
XRT_ANNO_MAX
```

XrtAttachType

Enumeration used to specify the type of text area attachment:

```
XRT_TEXT_ATTACH_PIXEL
XRT_TEXT_ATTACH_VALUE
XRT_TEXT_ATTACH_DATA
XRT_TEXT_ATTACH_DATA_VALUE
```

XrtAxis

Enumeration used to specify an axis. Used with *XrtGetValueLabel()* and *XrtSetValueLabel()*:

```
XRT_AXIS_X
XRT_AXIS_Y
XRT_AXIS_Y2
```

XrtBorder

Enumeration used by `XtNxrtBorder`, `XtNxrtHeaderBorder`, `XtNxrtFooterBorder`, `XtNxrtLegendBorder`, and `XtNxrtGraphBorder` and text areas:

```
XRT_BORDER_NONE
XRT_BORDER_PLAIN
XRT_BORDER_3D_OUT
XRT_BORDER_3D_IN
XRT_BORDER_ETCHED_IN
XRT_BORDER_ETCHED_OUT
XRT_BORDER_FRAME_IN
XRT_BORDER_FRAME_OUT
XRT_BORDER_BEVEL
XRT_BORDER_SHADOW
```

XrtCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtExposeCallback` and `XtNxrtResizeCallback` lists:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Window      window;
    Region      region;
    int         change;
} XrtCallbackStruct;
```

XrtColor

Platform specific type that is defined as *char **.

XrtDataStyle

The structure defining how a particular set of data will appear when graphed:

```
typedef struct {
    XrtLinePattern      lpat;   /* line pattern */
    XrtFillPattern      fpat;   /* fill pattern */
    XrtColor            color;  /* line color */
    int                 width;  /* line width */
    XrtPoint            point;  /* point style */
    XrtColor            pcolor; /* point color */
    int                 psize;  /* point size */
    unsigned long       res1;   /* reserved */
    unsigned long       res2;   /* reserved */
    unsigned long       resh;   /* reserved */
    unsigned long       ress;   /* reserved */
} XrtDataStyle;
```

XrtDisplay

Enumeration used to specify how to display individual sets in the data:

```
XRT_DISPLAY_SHOW
XRT_DISPLAY_HIDE
XRT_DISPLAY_EXCLUDE
```

XrtDrawPSColor

Enumeration used for specifying color-handling with *XrtDrawPS()*:

```
XRT_PS_COLOR_AS_IS
XRT_PS_COLOR_AUTO
XRT_PS_COLOR_MONO
```

XrtFillPattern

Enumeration of various fill patterns used within an `XrtDataStyle` structure:

```
XRT_FPAT_NONE
XRT_FPAT_SOLID
XRT_FPAT_25_PERCENT
XRT_FPAT_50_PERCENT
XRT_FPAT_75_PERCENT
XRT_FPAT_HORIZ_STRIPE
XRT_FPAT_VERT_STRIPE
XRT_FPAT_45_STRIPE
XRT_FPAT_135_STRIPE
XRT_FPAT_DIAG_HATCHED
XRT_FPAT_CROSS_HATCHED
```

XrtImage

Platform specific type that is defined to be an `XtPointer`.

Note that this type can be used in conjunction with `XrtPDSImage`, which is also defined to be an `XtPointer`.

XrtLinePattern

Enumeration of various line patterns, used within an `XrtDataStyle` structure:

```
XRT_LPAT_NONE
XRT_LPAT_SOLID
XRT_LPAT_LONG_DASH
XRT_LPAT_DOTTED
XRT_LPAT_SHORT_DASH
XRT_LPAT_LSL_DASH
XRT_LPAT_DASH_DOT
```

XrtMapCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtMapCallback` list:

```
typedef struct {
    int             reason;
    XEvent          *event;
    String          *params;
    Cardinal        nparams;
    XrtRegion       region;
    XrtMapResult    map;
} XrtMapCallbackStruct;
```

XrtMapResult

Structure used to pass information about mapped pixel coordinates:

```
typedef struct {
    int     pix_x, pix_y;
    int     yaxis;
    float   x, y;
} XrtMapResult;
```

XrtMarkerMethod

Enumeration used with the `XtNxrtXMarkerMethod` resource to select the X-marker positioning method:

```
XRT_MARKER_SET_POINT
XRT_MARKER_VALUE
```

XrtModifyCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtModifyCallback` list:

```
typedef struct {
    int         reason;
    XEvent      *event;
    String      *params;
    Cardinal    nparams;
    Boolean     doit;
} XrtModifyCallbackStruct;
```

XrtNumMethod

Enumeration used with the `XtNxrt[XYY2]NumMethod` resource to determine how to calculate axis annotation:

```
XRT_NUM_PRECISION
XRT_NUM_ROUND
```

XrtOriginPlacement

Enumeration used to select the way in which the default origin is chosen:

```
XRT_ORIGIN_AUTO
XRT_ORIGIN_ZERO
XRT_ORIGIN_MIN
XRT_ORIGIN_MAX
```

XrtPickCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtPickCallback` list:

```
typedef struct {
    int             reason;
    XEvent          *event;
    String          *params;
    Cardinal        nparams;
    XrtRegion       region;
    XrtPickResult   pick;
} XrtPickCallbackStruct;
```

XrtPickResult

Structure used to pass information about picked pixel coordinates:

```
typedef struct {
    int pix_x, pix_y;
    int dataset;
    int set, point;
    int distance;
} XrtPickResult;
```

XrtPieOrder

Enumeration of pie ordering styles:

```
XRT_PIEORDER_ASCENDING
XRT_PIEORDER_DESCENDING
XRT_PIEORDER_DATA_ORDER
```

XrtPieThresholdMethod

Enumeration of pie threshold methods:

```
XRT_PIE_SLICE_CUTOFF
XRT_PIE_PERCENTILE
```

XrtPoint

Enumeration of various point styles used in `XrtDataStyle`:

```
XRT_POINT_NONE
XRT_POINT_DOT
XRT_POINT_BOX
XRT_POINT_TRI
XRT_POINT_DIAMOND
XRT_POINT_STAR
XRT_POINT_VERT_LINE
XRT_POINT_HORIZ_LINE
XRT_POINT_CROSS
XRT_POINT_CIRCLE
XRT_POINT_SQUARE
XRT_POINT_INVERT_TRI
XRT_POINT_DIAG_CROSS
XRT_POINT_OPEN_TRI
XRT_POINT_OPEN_DIAMOND
XRT_POINT_OPEN_INVERT_TRI
```

XrtProperty

Enumeration of cross-platform names for XRT/graph resources, used with the *XrtGetPropString()*, *XrtGetValues()*, *XrtSetPropString()*, *XrtTextAreaSetPropString()*, *XrtTextAreaGetPropString()*, and *XrtSetValues()* procedures. For a list of the definition for each resource, see page 267.

XrtReason

Enumeration used to define the reason for a callback. Used with the `XtNxrtMapCallback`, `XtNxrtModifyCallback`, `XtNxrtPickCallback`, `XtNxrtRotateCallback`, `XtNxrtTransformCallback`, and `XtNxrtZoomAxisCallback` resources:

```
XRT_REASON_MODIFY_START
XRT_REASON_MODIFY_END
XRT_REASON_ROTATE
XRT_REASON_TRANSFORM
XRT_REASON_ZOOM_AXIS
XRT_REASON_MAP
XRT_REASON_PICK
```

XrtRegion

Enumeration of map or pick results. Returned by *XrtMap()* and *XrtPick()*:

```
XRT_RGN_NOWHERE
XRT_RGN_IN_GRAPH
XRT_RGN_IN_LEGEND
XRT_RGN_IN_FOOTER
XRT_RGN_IN_HEADER
```

XrtRotate

Enumeration used to define counter-clockwise rotation in degrees. Used with title and annotation rotation resources, also with text objects:

```
XRT_ROTATE_NONE
XRT_ROTATE_90
XRT_ROTATE_270
```

XrtRotateCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtRotateCallback` list:

```
typedef struct {
    int             reason;     // Read-only
    XEvent          *event;     // Read-only
    String          *params;    // Read-only
    Cardinal        nparams;    // Read-only
    int             rotation;
    int             inclination;
    Boolean         doit;
} XrtRotateCallbackStruct;
```

XrtShading

Used to control the shading method used with graphs displaying a 3D effect:

```
XRT_SHADING_COLOR
XRT_SHADING_DITHERED
XRT_SHADING_NONE
```

XrtTimeUnit

Enumeration used to define time units. Used with `XtNxrtTimeUnit`:

```
XRT_TMUNIT_SECONDS
XRT_TMUNIT_MINUTES
XRT_TMUNIT_HOURS
XRT_TMUNIT_DAYS
XRT_TMUNIT_WEEKS
XRT_TMUNIT_MONTHS
XRT_TMUNIT_YEARS
```

XrtTransformCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtTransformCallback` list:

```
typedef struct {
    int             reason;     // Read-only
    XEvent          *event;     // Read-only
    String          *params;    // Read-only
    Cardinal        nparams;    // Read-only
    Position        left_margin;
    Position        right_margin;
    Position        top_margin;
    Position        bottom_margin;
    Boolean         doit;
} XrtTransformCallbackStruct;
```

XrtType

Enumeration used to define graph types. Used with `XtNxrtType`:

```
XRT_TYPE_PLOT
XRT_TYPE_BAR
XRT_TYPE_PIE
XRT_TYPE_STACKING_BAR
XRT_TYPE_AREA
XRT_TYPE_HILO
XRT_TYPE_HILO_OPEN_CLOSE
XRT_TYPE_CANDLE
XRT_TYPE_BUBBLE
XRT_TYPE_POLAR
XRT_TYPE_RADAR
XRT_TYPE_FILLED_RADAR
```

XrtValueLabel

The structure defining a single Value-label for the X-, Y-, or Y2-axis. Used with the `XtNxrt[XYY2]Labels` resource.

```
typedef struct {
    float   value;
    char    *string;
} XrtValueLabel;
```

XrtZoomAxisCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrtZoomAxisCallback` list:

```
typedef struct {
    int         reason;     // Read-only
    XEvent      *event;     // Read-only
    String      *params;    // Read-only
    Cardinal    nparams;    // Read-only
    Boolean     reset;      // Read-only
    float       new_xaxis_min;
    float       new_xaxis_max;
    float       new_yaxis_min;
    float       new_yaxis_max;
    float       new_y2axis_min;
    float       new_y2axis_max;
    Boolean     doit;
} XrtZoomAxisCallbackStruct;
```

G

Resource Names, Types and Classes

This appendix lists the resource type and resource class for each XRT/graph resource. Underneath each resource name is listed its `XrtProperty` enumeration.

Name	Class	Type
XtNxrtAngleUnit XRT_ANGLE_UNIT	XtCXrtAngleUnit	XtRXrtAngleUnit
XtNxrtAxisBoundingBox XRT_AXIS_BOUNDING_BOX	XtCXrtDrawOption	XtRBoolean
XtNxrtAxisFont XRT_AXIS_FONT	XmCFontList	XmRFontList
XtNxrtBackgroundColor XRT_BACKGROUND_COLOR	XtCXrtBackgroundColor	XtRXrtColor
XtNxrtBarClusterOverlap XRT_BAR_CLUSTER_OVERLAP	XtCXrtBarClusterOverlap	XtRInt
XtNxrtBarClusterWidth XRT_BAR_CLUSTER_WIDTH	XtCXrtBarClusterWidth	XtRInt
XtNxrtBorder XRT_BORDER	XtCXrtBorder	XtRXrtBorder
XtNxrtBubbleMax XRT_BUBBLE_MAX	XtCXrtTypeBubbleBounds	XtRFloat
XtNxrtBubbleMethod XRT_BUBBLE_METHOD	XtCXrtBubbleMethod	XtRXrtBubbleMethod
XtNxrtBubbleMin XRT_BUBBLE_MIN	XtCXrtTypeBubbleBounds	XtRFloat
XtNxrtCandleComplex XRT_CANDLE_COMPLEX	XtCXrtCandleOption	XtRBoolean
XtNxrtColorFunc XRT_COLOR_FUNC	XtCXrtFunction	XtRFunction

Name	Class	Type
XtNxrtData XRT_DATA	XtCXrtData	XtRXrtData
XtNxrtData2 XRT_DATA2	XtCXrtData	XtRXrtData
XtNxrtDataAreaBackgroundColor XRT_DATA_AREA_BACKGROUND_COLOR	XtCXrtBackgroundColor	XtRXrtColor
XtNxrtDataAreaForegroundColor XRT_DATA_AREA_FOREGROUND_COLOR	XtCXrtForegroundColor	XtRXrtColor
XtNxrtDataAreaImage XRT_DATA_AREA_IMAGE	XtCXrtImage	XtRXrtImage
XtNxrtDataAreaImageLayout XRT_DATA_AREA_IMAGE_LAYOUT	XtCXrtImageLayout	XtRXrtImageLayout
XtNxrtDataStyles XRT_DATA_STYLES	XtCXrtDataStyles	XtRXrtDataStyles
XtNxrtDataStyles2 XRT_DATA_STYLES2	XtCXrtDataStyles	XtRXrtDataStyles
XtNxrtDataStylesUseDefault XRT_DATA_STYLES_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrtDataStyles2UseDefault XRT_DATA_STYLES2_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrtDebug XRT_DEBUG	XtCXrtDebug	XtRBoolean
XtNxrtDoubleBuffer XRT_DOUBLE_BUFFER	XtCXrtDoubleBuffer	XtRBoolean
XtNxrtExposeCallback [a]	XtCCallback	XtRCallback
XtNxrtExtraDefaultDataStyles XRT_EXTRA_DEFAULT_DATA_STYLES	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrtFooterAdjust XRT_FOOTER_ADJUST	XtCXrtAdjust	XtRXrtAdjust
XtNxrtFooterBackgroundColor XRT_FOOTER_BACKGROUND_COLOR	XtCXrtBackgroundColor	XtRXrtColor
XtNxrtFooterBorder XRT_FOOTER_BORDER	XtCXrtBorder	XtRXrtBorder
XtNxrtFooterBorderWidth XRT_FOOTER_BORDER_WIDTH	XtCXrtBorderWidth	XtRDimension
XtNxrtFooterFont XRT_FOOTER_FONT	XmCFontList	XmRFontList
XtNxrtFooterForegroundColor XRT_FOOTER_FOREGROUND_COLOR	XtCXrtForegroundColor	XtRXrtColor
XtNxrtFooterHeight XRT_FOOTER_HEIGHT	XtCXrtDimension	XtRShort

Name	Class	Type
XtNxrtFooterImage XRT_FOOTER_IMAGE	XtCXrtImage	XtRXrtImage
XtNxrtFooterImageLayout XRT_FOOTER_IMAGE_LAYOUT	XtCXrtImageLayout	XtRXrtImageLayout
XtNxrtFooterImageMinimumSize XRT_FOOTER_IMAGE_MINIMUM_SIZE	XtCXrtImageMinimumSize	XtRXBoolean
XtNxrtFooterStrings XRT_FOOTER_STRINGS	XtCXrtStrings	XtRXrtStrings
XtNxrtFooterWidth XRT_FOOTER_WIDTH	XtCXrtDimension	XtRShort
XtNxrtFooterX XRT_FOOTER_X	XtCXrtPosition	XtRShort
XtNxrtFooterXUseDefault XRT_FOOTER_X_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtFooterY XRT_FOOTER_Y	XtCXrtPosition	XtRShort
XtNxrtFooterYUseDefault XRT_FOOTER_Y_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtForegroundColor XRT_FOREGROUND_COLOR	XtCXrtForegroundColor	XtRXrtColor
XtNxrtFrontDataset XRT_FRONT_DATASET	XtCXrtDataset	XtRInt
XtNxrtGraph3dShading XRT_GRAPH_3D_SHADING	XtCXrtGraph3dShading	XtRXrtShading
XtNxrtGraphBackgroundColor XRT_GRAPH_BACKGROUND_COLOR	XtCXrtBackgroundColor	XtRXrtColor
XtNxrtGraphBorder XRT_GRAPH_BORDER	XtCXrtBorder	XtRXrtBorder
XtNxrtGraphBorderWidth XRT_GRAPH_BORDER_WIDTH	XtCXrtBorderWidth	XtRDimension
XtNxrtGraphDepth XRT_GRAPH_DEPTH	XtCXrtGraphDepth	XtRInt
XtNxrtGraphForegroundColor XRT_GRAPH_FOREGROUND_COLOR	XtCXrtForegroundColor	XtRXrtColor
XtNxrtGraphHeight XRT_GRAPH_HEIGHT	XtCXrtDimension	XtRShort
XtNxrtGraphHeightUseDefault XRT_GRAPH_HEIGHT_USE_DEFAULT	XtCXrtDimensionUseDefault	XtRBoolean
XtNxrtGraphImage XRT_GRAPH_IMAGE	XtCXrtImage	XtRXrtImage

Name	Class	Type
XtNxrtGraphImageLayout XRT_GRAPH_IMAGE_LAYOUT	XtCXrtImageLayout	XtRXrtImageLayout
XtNxrtGraphInclination XRT_GRAPH_INCLINATION	XtCXrtGraphInclination	XtRInt
XtNxrtGraphMarginBottom XRT_GRAPH_MARGIN_BOTTOM	XtCXrtGraphMargin	XtRShort
XtNxrtGraphMarginBottomUseDefault XRT_GRAPH_MARGIN_BOTTOM_USE_DEFAULT	XtCXrtGraphMarginUseDefault	XtRBoolean
XtNxrtGraphMarginLeft XRT_GRAPH_MARGIN_LEFT	XtCXrtGraphMargin	XtRShort
XtNxrtGraphMarginLeftUseDefault XRT_GRAPH_MARGIN_LEFT_USE_DEFAULT	XtCXrtGraphMarginUseDefault	XtRBoolean
XtNxrtGraphMarginRight XRT_GRAPH_MARGIN_RIGHT	XtCXrtGraphMargin	XtRShort
XtNxrtGraphMarginRightUseDefault XRT_GRAPH_MARGIN_RIGHT_USE_DEFAULT	XtCXrtGraphMarginUseDefault	XtRBoolean
XtNxrtGraphMarginTop XRT_GRAPH_MARGIN_TOP	XtCXrtGraphMargin	XtRShort
XtNxrtGraphMarginTopUseDefault XRT_GRAPH_MARGIN_TOP_USE_DEFAULT	XtCXrtGraphMarginUseDefault	XtRBoolean
XtNxrtGraphRotation XRT_GRAPH_ROTATION	XtCXrtGraphRotation	XtRInt
XtNxrtGraphShowOutlines XRT_GRAPH_SHOW_OUTLINES	XtCXrtGraphOption	XtRBoolean
XtNxrtGraphWidth XRT_GRAPH_WIDTH	XtCXrtDimension	XtRShort
XtNxrtGraphWidthUseDefault XRT_GRAPH_WIDTH_USE_DEFAULT	XtCXrtDimensionUseDefault	XtRBoolean
XtNxrtGraphX XRT_GRAPH_X	XtCXrtPosition	XtRShort
XtNxrtGraphXUseDefault XRT_GRAPH_X_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtGraphY XRT_GRAPH_Y	XtCXrtPosition	XtRShort
XtNxrtGraphYUseDefault XRT_GRAPH_Y_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtHeaderAdjust XRT_HEADER_ADJUST	XtCXrtAdjust	XtRXrtAdjust
XtNxrtHeaderBackgroundColor XRT_HEADER_BACKGROUND_COLOR	XtCXrtBackgroundColor	XtRXrtColor

Name	Class	Type
XtNxrtHeaderBorder XRT_HEADER_BORDER	XtCXrtBorder	XtRXrtBorder
XtNxrtHeaderBorderWidth XRT_HEADER_BORDER_WIDTH	XtCXrtBorderWidth	XtRDimension
XtNxrtHeaderFont XRT_HEADER_FONT	XmCFontList	XmRFontList
XtNxrtHeaderForegroundColor XRT_HEADER_FOREGROUND_COLOR	XtCXrtForegroundColor	XtRXrtColor
XtNxrtHeaderHeight XRT_HEADER_HEIGHT	XtCXrtDimension	XtRShort
XtNxrtHeaderImage XRT_HEADER_IMAGE	XtCXrtImage	XtRXrtImage
XtNxrtHeaderImageLayout XRT_HEADER_IMAGE_LAYOUT	XtCXrtImageLayout	XtRXrtImageLayout
XtNxrtHeaderImageMinimumSize XRT_HEADER_IMAGE_MINIMUM_SIZE	XtCXrtImageMinimumSize	XtRBoolean
XtNxrtHeaderStrings XRT_HEADER_STRINGS	XtCXrtStrings	XtRXrtStrings
XtNxrtHeaderWidth XRT_HEADER_WIDTH	XtCXrtDimension	XtRShort
XtNxrtHeaderX XRT_HEADER_X	XtCXrtPosition	XtRShort
XtNxrtHeaderXUseDefault XRT_HEADER_X_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtHeaderY XRT_HEADER_Y	XtCXrtPosition	XtRShort
XtNxrtHeaderYUseDefault XRT_HEADER_Y_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtHiLoCloseShow XRT_HILO_CLOSE_SHOW	XtCXrtHiLoOption	XtRBoolean
XtNxrtHiLoOpenCloseFullWidth XRT_HILO_OPEN_CLOSE_FULL_WIDTH	XtCXrtHiLoOption	XtRBoolean
XtNxrtHiLoOpenShow XRT_HILO_OPEN_SHOW	XtCXrtHiLoOption	XtRBoolean
XtNxrtImage XRT_IMAGE	XtCXrtImage	XtRXrtImage
XtNxrtImageLayout XRT_IMAGE_LAYOUT	XtCXrtImageLayout	XtRXrtImageLayout
XtNxrtInvertOrientation XRT_INVERT_ORIENTATION	XtCXrtDrawOption	XtRBoolean

Name	Class	Type
XtNxrtIsStacked XRT_IS_STACKED	XtCXrtIsStacked	XtRBoolean
XtNxrtIsStacked2 XRT_IS_STACKED2	XtCXrtIsStacked	XtRBoolean
XtNxrtLegendAnchor XRT_LEGEND_ANCHOR	XtCXrtAnchor	XtRXrtAnchor
XtNxrtLegendBackgroundColor XRT_LEGEND_BACKGROUND_COLOR	XtCXrtBackgroundColor	XtRXrtColor
XtNxrtLegendBorder XRT_LEGEND_BORDER	XtCXrtBorder	XtRXrtBorder
XtNxrtLegendBorderWidth XRT_LEGEND_BORDER_WIDTH	XtCXrtBorderWidth	XtRDimension
XtNxrtLegendFont XRT_LEGEND_FONT	XmCFontList	XmRFontList
XtNxrtLegendForegroundColor XRT_LEGEND_FOREGROUND_COLOR	XtCXrtForegroundColor	XtRXrtColor
XtNxrtLegendHeight XRT_LEGEND_HEIGHT	XtCXrtDimension	XtRShort
XtNxrtLegendImage XRT_LEGEND_IMAGE	XtCXrtImage	XtRXrtImage
XtNxrtLegendImageLayout XRT_LEGEND_IMAGE_LAYOUT	XtCXrtImageLayout	XtRXrtImageLayout
XtNxrtLegendOrientation XRT_LEGEND_ORIENTATION	XtCXrtAlign	XtRXrtAlign
XtNxrtLegendReversed XRT_LEGEND_REVERSED	XtCXrtDrawOption	XtRBoolean
XtNxrtLegendShow XRT_LEGEND_SHOW	XtCXrtDrawOption	XtRBoolean
XtNxrtLegendTitle XRT_LEGEND_TITLE	XtCXrtTitle	XtRXrtString
XtNxrtLegendWidth XRT_LEGEND_WIDTH	XtCXrtDimension	XtRShort
XtNxrtLegendX XRT_LEGEND_X	XtCXrtPosition	XtRShort
XtNxrtLegendXUseDefault XRT_LEGEND_X_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtLegendY XRT_LEGEND_Y	XtCXrtPosition	XtRShort
XtNxrtLegendYUseDefault XRT_LEGEND_Y_USE_DEFAULT	XtCXrtPositionUseDefault	XtRBoolean
XtNxrtMapCallback [a]	XtCCallback	XtRCallback

Name	Class	Type
XtNxrtMarkerDataset XRT_MARKER_DATASET	XtCXrtDataset	XtRInt
XtNxrtMarkerDataStyle XRT_MARKER_DATA_STYLE	XtCXrtDataStyle	XtRXrtDataStyle
XtNxrtMarkerDataStyleUseDefault XRT_MARKER_DATA_STYLE_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrtModifyCallback [a]	XtCCallback	XtRCallback
XtNxrtOtherDataStyle XRT_OTHER_DATA_STYLE	XtCXrtDataStyle	XtRXrtDataStyle
XtNxrtOtherDataStyleUseDefault XRT_OTHER_DATA_STYLE_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrtOtherLabel XRT_OTHER_LABEL	XtCXrtOtherLabel	XtRXrtString
XtNxrtPickCallback [a]	XtCCallback	XtRCallback
XtNxrtPieMergeMissingSlices XRT_PIE_MERGE_MISSING_SLICES	XtCXrtPieSlices	XtRBoolean
XtNxrtPieMinSlices XRT_PIE_MIN_SLICES	XtCXrtPieMinSlices	XtRInt
XtNxrtPieOrder XRT_PIE_ORDER	XtCXrtPieOrder	XtRXrtPieOrder
XtNxrtPieStartAngle XRT_PIE_START_ANGLE	XtCXrtPieStartAngle	XtRFloat
XtNxrtPieThresholdMethod XRT_PIE_THRESHOLD_METHOD	XtCXrtPieThresholdMethod	XtRXrtPieThresholdMethod
XtNxrtPieThresholdValue XRT_PIE_THRESHOLD_VALUE	XtCXrtPieThresholdValue	XtRFloat
XtNxrtPointLabels XRT_POINT_LABELS	XtCXrtPointLabels	XtRXrtStrings
XtNxrtPointLabels2 XRT_POINT_LABELS2	XtCXrtPointLabels	XtRXrtStrings
XtNxrtPolarAxisAllowNegatives XRT_POLAR_AXIS_ALLOW_NEGATIVES	XtCXrtPolarOption	XtRBoolean
XtNxrtPolarHalfRange XRT_POLAR_HALF_RANGE	XtCXrtPolarOption	XtRBoolean
XtNxrtRepaint XRT_REPAINT	XtCXrtRepaint	XtRBoolean
XtNxrtResizeCallback [a]	XtCCallback	XtRCallback
XtNxrtRotateCallback[a]	XtCCallback	XtRCallback
XtNxrtSetLabels XRT_SET_LABELS	XtCXrtSetLabels	XtRXrtStrings

Name	Class	Type
XtNxrtSetLabels2 XRT_SET_LABELS2	XtCXrtSetLabels	XtRXrtStrings
XtNxrtTimeBase XRT_TIME_BASE	XtCXrtTimeBase	XtRLong
XtNxrtTimeFormat XRT_TIME_FORMAT	XtCXrtTimeFormat	XrRXrtString
XtNxrtTimeFormatUseDefault XRT_TIME_FORMAT_USE_DEFAULT	XtCXrtTimeFormatUseDefault	XtRBoolean
XtNxrtTimeUnit XRT_TIME_UNIT	XtCXrtTimeUnit	XtRXrtTimeUnit
XtNxrtTransformCallback [a]	XtCCallback	XtRCallback
XtNxrtType XRT_TYPE	XtCXrtType	XtRXrtType
XtNxrtType2 XRT_TYPE2	XtCXrtType	XtRXrtType
XtNxrt[XYY2]AnnoPlacement XRT_[XYY2]ANNO_PLACEMENT	XtCXrtAnnoPlacement	XtRXrtAnnoPlacement
XtNxrt[XYY2]AnnotationMethod XRT_[XYY2]ANNOTATION_METHOD	XtCXrtAnnotationMethod	XtRXrtAnnotationMethod
XtNxrt[XYY2]AnnotationRotation XRT_[XYY2]ANNOTATION_ROTATION	XtCXrtRotation	XtRXrtRotate
XtNxrt[XYY2]AxisDataStyle XRT_[XYY2]AXIS_DATA_STYLE	XtCXrtDataStyle	XtRXrtDataStyle
XtNxrt[XYY2]AxisDataStyleUseDefault XRT_[XYY2]AXIS_DATA_STYLE_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrt[XYY2]AxisLabelFormat XRT_[XYY2]AXIS_LABEL_FORMAT	XtCXrtAxisLabelFormat	XtRXrtString
XtNxrt[XYY2]AxisLogarithmic XRT_[XYY2]AXIS_LOGARITHMIC	XtCXrtAxisLogarithmic	XtRBoolean
XtNxrt[XYY2]AxisMax XRT_[XYY2]AXIS_MAX	XtCXrtAxisBounds	XtRFloat
XtNxrt[XYY2]AxisMaxUseDefault XRT_[XYY2]AXIS_MAX_USE_DEFAULT	XtCXrtAxisBoundsUseDefault	XtRBoolean
XtNxrt[XYY2]AxisMin XRT_[XYY2]AXIS_MIN	XtCXrtAxisBounds	XtRFloat
XtNxrt[XYY2]AxisMinUseDefault XRT_[XYY2]AXIS_MIN_USE_DEFAULT	XtCXrtAxisBoundsUseDefault	XtRBoolean
XtNxrt[XYY2]AxisReversed XRT_[XYY2]AXIS_REVERSED	XtCXrtAxisReversed	XtRBoolean
XtNxrt[XYY2]AxisShow XRT_[XYY2]AXIS_SHOW	XtCXrtAxisShow	XtRBoolean

Name	Class	Type
XtNxrt[XYY2]Grid XRT_[XYY2]GRID	XtCXrtGrid	XtRFloat
XtNxrt[XYY2]GridDataStyle XRT_[XYY2]GRID_DATA_STYLE	XtCXrtDataStyle	XtRXrtDataStyle
XtNxrt[XYY2]GridDataStyleUseDefault XRT_[XYY2]GRID_DATA_STYLE_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrt[XYY2]GridUseDefault XRT_[XYY2]GRID_USE_DEFAULT	XtCXrtGridUseDefault	XtRBoolean
XtNxrt[XYY2]Labels XRT_[XYY2]LABELS	XtCXrtValueLabels	XtRXrtValueLabels
XtNxrt[XY]Marker XRT_[XY]MARKER	XtCXrtMarker	XtRFloat
XtNxrt[XY]MarkerDataStyle XRT_[XY]MARKER_DATA_STYLE	XtCXrtDataStyle	XtRXrtDataStyle
XtNxrt[XY]MarkerDataStyleUseDefault XRT_[XY]MARKER_DATA_STYLE_USE_DEFAULT	XtCXrtDataStylesUseDefault	XtRBoolean
XtNxrtXMarkerMethod XRT_XMARKER_METHOD	XtCXrtMarkerMethod	XtRXrtMarkerMethod
XtNxrtXMarkerPoint XRT_XMARKER_POINT	XtCXrtMarkerPoint	XtRInt
XtNxrtXMarkerSet XRT_XMARKER_SET	XtCXrtMarkerSet	XtRInt
XtNxrt[XY]MarkerShow XRT_[XY]MARKER_SHOW	XtCXrtMarkerShow	XtRBoolean
XtNxrt[XYY2]Max XRT_[XYY2]MAX	XtCXrtAxisBounds	XtRFloat
XtNxrt[XYY2]MaxUseDefault XRT_[XYY2]MAX_USE_DEFAULT	XtCXrtAxisBoundsUseDefault	XtRBoolean
XtNxrt[XYY2]Min XRT_[XYY2]MIN	XtCXrtAxisBounds	XtRFloat
XtNxrt[XYY2]MinUseDefault XRT_[XYY2]MIN_USE_DEFAULT	XtCXrtAxisBoundsUseDefault	XtRBoolean
XtNxrt[XYY2]Num XRT_[XYY2]NUM	XtCXrtAxisParams	XtRFloat
XtNxrt[XYY2]NumMethod XRT_[XYY2]NUM_METHOD	XtCXrtNumMethod	XtRXrtNumMethod
XtNxrt[XYY2]NumUseDefault XRT_[XYY2]NUM_USE_DEFAULT	XtCXrtAxisParamsUseDefault	XtRBoolean
XtNxrt[XYY2]Origin XRT_[XYY2]ORIGIN	XtCXrtAxisParams	XtRFloat

Name	Class	Type
XtNxrtXOriginBase XRT_XORIGIN_BASE	XtCXrtAxisParams	XtRFloat
XtNxrt[XYY2]OriginPlacement XRT_[XYY2]ORIGIN_PLACEMENT	XtCXrtOriginPlacement	XtRXrtOriginPlacement
XtNxrt[XYY2]OriginUseDefault XRT_[XYY2]ORIGIN_USE_DEFAULT	XtCXrtAxisParamsUseDefault	XtRBoolean
XtNxrt[XYY2]Precision XRT_[XYY2]PRECISION	XtCXrtPrecision	XtRInt
XtNxrt[XYY2]PrecisionUseDefault XRT_[XYY2]PRECISION_USE_DEFAULT	XtCXrtPrecisionUseDefault	XtRBoolean
XtNxrt[XYY2]Tick XRT_[XYY2]TICK	XtCXrtAxisParams	XtRFloat
XtNxrt[XYY2]TickUseDefault XRT_[XYY2]TICK_USE_DEFAULT	XtCXrtAxisParamsUseDefault	XtRBoolean
XtNxrt[XYY2]Title XRT_[XYY2]TITLE	XtCXrtTitle	XtRXrtString
XtNxrt[XYY2]TitleRotation XRT_[XYY2]TITLE_ROTATION	XtCXrtRotation	XtRXrtRotate
XtNxrt[YY2]AnnotationAngle XRT_[YY2]ANNOTATION_ANGLE	XtCXrtAxisParams	XtRFloat
XtNxrt[YY2]AnnotationAngleUseDefault XRT_[YY2]ANNOTATION_ANGLE_USE_DEFAULT	XtCXrtAxisParamsUseDefault	XtRBoolean
XtNxrt[YY2]Axis100Percent XRT_[YY2]AXIS_100_PERCENT	XtCXrtAxisParams	XtRBoolean
XtNxrtYAxisConst XRT_YAXIS_CONST	XtCXrtAxisConst	XtRFloat
XtNxrtYAxisMult XRT_YAXIS_MULT	XtCXrtAxisMult	XtRFloat
XtNxrtZoomAxisCallback [a]	XtCCallback	XtRCallback

a. This resource is not used on Windows so no cross-platform name is defined.

H

Resource File and GUI Builder Usage

Syntax
Example Resource File

This appendix lists the syntax for setting XRT/graph resources in resource files, using GUI builder tools, and specifies the format of the strings refused by *XrtGetPropString()* and set by *XrtSetPropString()*. It also lists a sample resource file. XRT/graph only defines resource converters for those data types which are not already defined by Xt and Motif.

All XRT/graph resources except `UseDefault` resources may be defined in a resource file.

H.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 `XRT_` 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 XRT_BORDER_SHADOW to be used for XtNxrtGraphBorder:

```
*.XtXrtGraph.xrtGraphBorder: BorderShadow
*.XtXrtGraph.xrtGraphBorder: BORDERSHADOW
*.XtXrtGraph.xrtGraphBorder: bordershadow
```

Data

Resources of type XtRXrtData are specified with any valid filename. The resource converter opens the file and calls *XrtDataCreateFromFile()*. Any error messages are written to *stderr*. An example:

```
*.XtXrtGraph.xrtData: data/net_data.dat
```

Data Style

One data style (resources of type XtRXrtDataStyle) is specified as a parenthesized list of seven values, in the order listed: line pattern, fill pattern, color, line thickness, point style, point color, and point size. The following example is the default style for the "other" pie slice:

```
*.bar.xrtOtherDataStyle: (LpatSolid FpatSolid \
        "yellow" 1 PointNone "black" 7 )
```

List of Data Styles

Resources of type XtRXrtDataStyles are specified as a parenthesized list, with each member specifying a complete data style (XtRXrtDataStyle). For example:

```
*.bar.xrtDataStyles: ((LpatSolid FpatSolid \
        "#6CCC1D" 3 PointTri "Dark Orchid" 9 )\
    (LpatSolid FpatSolid "#5A8DCC" 3 PointBox \
        "Indian Red" 9 )\
    (LpatSolid FpatSolid "LimeGreen" 1 PointDot \
        "medium blue" 7 )\
    (LpatSolid FpatSolid "Medium Turquoise" 1 \
        PointDiamond "red" 7 )\
    (LpatSolid FpatSolid "blue" 1 PointStar \
        "red" 7 ))
*.bar.xrtDataStyles2: ()
```

Images

Resources of type XtRXrtImage are specified with any valid filename. Filenames are checked for file type and are then converted to the proper image type if it's a default image type or an image type registered by the user. An example:

```
*.XtXrtGraph.xrtLegendImage: $XRTHOME/src/icons/shading.xpm
```

String Values

If a string (resources of type XtRXrtString) to be specified includes spaces, it must be enclosed in double quotes ("). The resource converter removes double quotes. To specify a double quote as part of a string, it must be preceded with another double quote ("").

List of Strings

Resources of type `XtRXrtStrings` are specified as a parenthesized list of strings (`XtRXrtString`). To specify an empty list, use a single pair of parentheses. For example:

```
*.xrtSetLabels: (Eggs "Ham and Cheese" Rye)
*.xrtPointLabels: ()
```

Value-labels

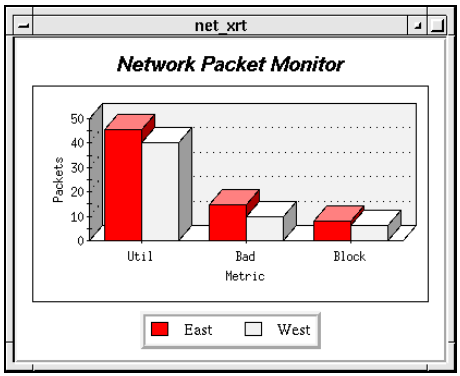
Resources of type `XtRXrtValueLabels` are specified as a parenthesized list, with each member of the list specifying a complete Value-label within a set of parentheses. The label string is contained within quotes. For example:

```
*.XtXrtGraph.xrtXLabels: (\
        (2.3 "Cold") \
        (3.5 "Warm") \
        (5.2 "Hot") \
    )
```

The converter sorts all Value-labels in increasing order by value. Duplicate entries are removed in arbitrary order. To specify an empty list, use a single pair of parentheses.

H.2 Example Resource File

The following resource file defines the graph shown below:




```
*.net.xrtAxisBoundingBox: True
*.net.xrtAxisFont: fixed
*.net.xrtBackgroundColor: "White"
*.net.xrtBarClusterOverlap: 100
*.net.xrtBarClusterWidth: 70
*.net.xrtDataAreaBackgroundColor: "grey95"
*.net.xrtDataStyles: ((LpatSolid FpatSolid "red" 1 PointTri "magenta" 7 )\
 (LpatSolid FpatSolid "grey95" 1 PointBox "green" 7 ))
*.net.xrtDataStyles2: ()
*.net.xrtDoubleBuffer: True
*.net.xrtFooterAdjust: AdjustCenter
*.net.xrtFooterBackgroundColor: ""
*.net.xrtFooterBorder: BorderNone
*.net.xrtFooterBorderWidth: 2
*.net.xrtFooterFont: fixed
*.net.xrtFooterForegroundColor: ""
*.net.xrtFooterStrings: ()
*.net.xrtFooterX: 198
*.net.xrtFooterY: 298
*.net.xrtForegroundColor: "Black"
*.net.xrtFrontDataset: 1
*.net.xrtGraphBackgroundColor: ""
*.net.xrtGraphBorder: BorderPlain
*.net.xrtGraphBorderWidth: 1
*.net.xrtGraphDepth: 10
*.net.xrtGraphForegroundColor: ""
*.net.xrtGraphHeight: 199
*.net.xrtGraphInclination: 30
*.net.xrtGraphMarginBottom: 55
*.net.xrtGraphMarginLeft: 51
*.net.xrtGraphMarginRight: 10
*.net.xrtGraphMarginTop: 15
*.net.xrtGraphRotation: 30
*.net.xrtGraphWidth: 364
*.net.xrtGraphX: 16
*.net.xrtGraphY: 44
```

```
*.net.xrtHeaderAdjust: AdjustCenter
*.net.xrtHeaderBackgroundColor: ""
*.net.xrtHeaderBorder: BorderNone
*.net.xrtHeaderBorderWidth: 2
*.net.xrtHeaderFont: -adobe-helvetica-bold-o-normal--18-180-75-75-p-104-iso8859-1
*.net.xrtHeaderForegroundColor: ""
*.net.xrtHeaderStrings: ("Network Packet Monitor")
*.net.xrtHeaderX: 94
*.net.xrtHeaderY: 15
*.net.height: 298
*.net.xrtInvertOrientation: False
*.net.xrtLegendAnchor: AnchorSouth
*.net.xrtLegendBackgroundColor: ""
*.net.xrtLegendBorder: BorderEtchedOut
*.net.xrtLegendBorderWidth: 5
*.net.xrtLegendFont: -adobe-times-medium-r-normal--14-140-75-75-p-74-iso8859-1
*.net.xrtLegendForegroundColor: ""
*.net.xrtLegendOrientation: AlignHorizontal
*.net.xrtLegendShow: True
*.net.xrtLegendX: 115
*.net.xrtLegendY: 251
*.net.xrtMarkerDataStyle: (LpatShortDash FpatSolid "black" 1 PointNone "black" 7 )
*.net.xrtOtherDataStyle: (LpatSolid FpatSolid "yellow" 1 PointNone "black" 7 )
*.net.xrtOtherLabel: "Other"
*.net.xrtPieMinSlices: 1
*.net.xrtPieOrder: PieOrderDataOrder
*.net.xrtPieThresholdMethod: PieSliceCutoff
*.net.xrtPieThresholdValue: 0.000000
*.net.xrtPointLabels: ("Util"\
 "Bad"\
 "Block")
*.net.xrtPointLabels2: ()
*.net.xrtSetLabels: ("East"\
 "West")
*.net.xrtSetLabels2: ()
*.net.xrtTimeBase: 0
*.net.xrtTimeFormat: ""
*.net.xrtTimeUnit: TmunitSeconds
*.net.xrtType: TypeBar
*.net.xrtType2: TypeBar
*.net.width: 396
*.net.xrtXAnnoPlacement: AnnoAuto
*.net.xrtXAnnotationMethod: AnnoPointLabels
*.net.xrtXAnnotationRotation: RotateNone
*.net.xrtXAxisLogarithmic: False
*.net.xrtXAxisMax: 1.000000
*.net.xrtXAxisMin: 0.000000
*.net.xrtXAxisReversed: False
*.net.xrtXAxisShow: True
*.net.xrtXGrid: 0.000000
*.net.xrtXGridDataStyle: (LpatDotted FpatSolid "Black" 1 PointNone "black" 7 )
*.net.xrtXLabels: ()
*.net.xrtXMax: 0.000000
*.net.xrtXMin: 0.000000
*.net.xrtXNum: 1.000000
*.net.xrtXNumMethod: NumRound
*.net.xrtXOrigin: 0.000000
*.net.xrtXOriginPlacement: OriginAuto
*.net.xrtXPrecision: 0
*.net.xrtXTick: 0.500000
```

```
*.net.xrtXTitle: "Metric"
*.net.xrtXTitleRotation: RotateNone
*.net.xrtY2AnnotationMethod: AnnoValues
*.net.xrtY2AnnotationRotation: RotateNone
*.net.xrtY2AxisLogarithmic: False
*.net.xrtY2AxisMax: 340282346638528859811704183484516925440.000000
*.net.xrtY2AxisMin: 340282346638528859811704183484516925440.000000
*.net.xrtY2AxisShow: True
*.net.xrtY2Labels: ()
*.net.xrtY2Max: 340282346638528859811704183484516925440.000000
*.net.xrtY2Min: 340282346638528859811704183484516925440.000000
*.net.xrtY2Num: 340282346638528859811704183484516925440.000000
*.net.xrtY2NumMethod: NumRound
*.net.xrtY2Precision: 27241
*.net.xrtY2Tick: 340282346638528859811704183484516925440.000000
*.net.xrtY2Title: ""
*.net.xrtYAnnoPlacement: AnnoAuto
*.net.xrtYAnnotationMethod: AnnoValues
*.net.xrtYAnnotationRotation: RotateNone
*.net.xrtY2AxisReversed: False
*.net.xrtYAxisConst: 0.000000
*.net.xrtYAxisLogarithmic: False
*.net.xrtYAxisMax: 50.000000
*.net.xrtYAxisMin: 0.000000
*.net.xrtYAxisMult: 0.000000
*.net.xrtYAxisReversed: False
*.net.xrtYAxisShow: True
*.net.xrtYGrid: 10.000000
*.net.xrtYGridDataStyle: (LpatDotted FpatSolid "Black" 1 PointNone "black" 7 )
*.net.xrtYLabels: ()
*.net.xrtYMax: 45.599998
*.net.xrtYMin: 6.200000
*.net.xrtYNum: 10.000000
*.net.xrtYNumMethod: NumRound
*.net.xrtYOrigin: 0.000000
*.net.xrtYOriginPlacement: OriginAuto
*.net.xrtYPrecision: -1
*.net.xrtYTick: 5.000000
*.net.xrtYTitle: "Packets"
*.net.xrtY2TitleRotation: RotateNone
*.net.xrtYTitleRotation: Rotate90
```

I

UIL Arguments

This appendix provides an alphabetical listing of XRT/graph's UIL arguments and their data types. A second table provides XRT/graph UIL reasons. For more information on using XRT/graph with UIL, see section 7.10 on page 141.

UIL Argument Name	Argument Type
XtNxrtAngleUnit	integer
XtNxrtAxisBoundingBox	boolean
XtNxrtAxisFont	font_table
XtNxrtBackgroundColor	string
XtNxrtBarClusterOverlap	integer
XtNxrtBarClusterWidth	integer
XtNxrtBorder	integer
XtNxrtBubbleMax	float
XtNxrtBubbleMethod	XrtBubbleMethod
XtNxrtBubbleMin	float
XtNxrtCandleComplex	boolean
XtNxrtDataAreaBackgroundColor	string
XtNxrtDataAreaForegroundColor	string
XtNxrtDataAreaImage	XrtImage
XtNxrtDataAreaImageLayout	integer
XtNxrtDataStyles2UseDefault	boolean
XtNxrtDataStylesUseDefault	boolean
XtNxrtDebug	boolean

UIL Argument Name	Argument Type
XtNxrtDoubleBuffer	boolean
XtNxrtExtraDefaultDataStyles	boolean
XtNxrtFooterAdjust	integer
XtNxrtFooterBackgroundColor	string
XtNxrtFooterBorder	integer
XtNxrtFooterBorderWidth	integer
XtNxrtFooterFont	font_table
XtNxrtFooterForegroundColor	string
XtNxrtFooterImage	XrtImage
XtNxrtFooterImageLayout	integer
XtNxrtFooterImageMinimumSize	boolean
XtNxrtFooterX	integer
XtNxrtFooterXUseDefault	boolean
XtNxrtFooterY	integer
XtNxrtFooterYUseDefault	boolean
XtNxrtForegroundColor	string
XtNxrtFrontDataset	integer
XtNxrtGraph3dShading	integer
XtNxrtGraphBackgroundColor	string
XtNxrtGraphBorder	integer
XtNxrtGraphBorderWidth	integer
XtNxrtGraphDepth	integer
XtNxrtGraphForegroundColor	string
XtNxrtGraphHeight	integer
XtNxrtGraphHeightUseDefault	boolean
XtNxrtGraphImage	XrtImage
XtNxrtGraphImageLayout	integer
XtNxrtGraphInclination	integer
XtNxrtGraphMarginBottom	integer
XtNxrtGraphMarginBottomUseDef [a]	boolean
XtNxrtGraphMarginLeft	integer
XtNxrtGraphMarginLeftUseDefault	boolean

UIL Argument Name	Argument Type
XtNxrtGraphMarginRight	integer
XtNxrtGraphMarginRightUseDef [a]	boolean
XtNxrtGraphMarginTop	integer
XtNxrtGraphMarginTopUseDefault	boolean
XtNxrtGraphRotation	integer
XtNxrtGraphShowOutlines	boolean
XtNxrtGraphWidth	integer
XtNxrtGraphWidthUseDefault	boolean
XtNxrtGraphX	integer
XtNxrtGraphXUseDefault	boolean
XtNxrtGraphY	integer
XtNxrtGraphYUseDefault	boolean
XtNxrtHeaderAdjust	integer
XtNxrtHeaderBackgroundColor	string
XtNxrtHeaderBorder	integer
XtNxrtHeaderBorderWidth	integer
XtNxrtHeaderFont	font_table
XtNxrtHeaderForegroundColor	string
XtNxrtHeaderImage	XrtImage
XtNxrtHeaderImageLayout	integer
XtNxrtHeaderImageMinimumSize	boolean
XtNxrtHeaderX	integer
XtNxrtHeaderXUseDefault	boolean
XtNxrtHeaderY	integer
XtNxrtHeaderYUseDefault	boolean
XtNxrtHiLoCloseShow	boolean
XtNxrtHiLoOpenCloseFullWidth	boolean
XtNxrtHiLoOpenShow	boolean
XtNxrtImage	XrtImage
XtNxrtImageLayout	integer
XtNxrtInvertOrientation	boolean
XtNxrtIsStacked	boolean

UIL Argument Name	Argument Type
XtNxrtIsStacked2	boolean
XtNxrtLegendAnchor	integer
XtNxrtLegendBackgroundColor	string
XtNxrtLegendBorder	integer
XtNxrtLegendBorderWidth	integer
XtNxrtLegendFont	font_table
XtNxrtLegendForegroundColor	string
XtNxrtLegendImage	XrtImage
XtNxrtLegendImageLayout	integer
XtNxrtLegendOrientation	integer
XtNxrtLegendReversed	boolean
XtNxrtLegendShow	boolean
XtNxrtLegendTitle	string
XtNxrtLegendX	integer
XtNxrtLegendXUseDefault	boolean
XtNxrtLegendY	integer
XtNxrtLegendYUseDefault	boolean
XtNxrtMarkerDataStyleUseDefault	boolean
XtNxrtMarkerDataset	integer
XtNxrtOtherDataStyleUseDefault	boolean
XtNxrtOtherLabel	string
XtNxrtPieMergeMissingSlices	boolean
XtNxrtPieMinSlices	integer
XtNxrtPieOrder	integer
XtNxrtPieStartAngle	float
XtNxrtPieThresholdMethod	integer
XtNxrtPieThresholdValue	single_float
XtNxrtPolarAxisAllowNegatives	boolean
XtNxrtPolarHalfRange	boolean
XtNxrtRepaint	boolean
XtNxrtTimeBase	integer
XtNxrtTimeFormat	string

UIL Argument Name	Argument Type
XtNxrtTimeFormatUseDefault	boolean
XtNxrtTimeUnit	integer
XtNxrtType	integer
XtNxrtType2	integer
XtNxrt[XYY2]AnnoPlacement	integer
XtNxrt[XYY2]AnnotationMethod	integer
XtNxrt[XYY2]AnnotationRotation	integer
XtNxrt[XYY2]AxisDataStyle	XrtDataStyle
XtNxrt[XYY2]AxisDataStyleUseDefault	boolean
XtNxrt[XYY2]AxisLabelFormat	string
XtNxrt[XYY2]AxisLogarithmic	boolean
XtNxrt[XYY2]AxisMax	single_float
XtNxrt[XYY2]AxisMaxUseDefault	boolean
XtNxrt[XYY2]AxisMin	single_float
XtNxrt[XYY2]AxisMinUseDefault	boolean
XtNxrt[XYY2]AxisReversed	boolean
XtNxrt[YY2]XAxisShow	boolean
XtNxrt[XYY2]Grid	single_float
XtNxrt[XYY2]GridDataStyleUseDefault	boolean
XtNxrt[XYY2]GridUseDefault	boolean
XtNxrt[XY]Marker	single_float
XtNxrt[XY]MarkerDataStyleUseDef [a]	boolean
XtNxrtXMarkerMethod	integer
XtNxrtXMarkerPoint	integer
XtNxrtXMarkerSet	integer
XtNxrt[XY]MarkerShow	boolean
XtNxrt[XYY2]Max	single_float
XtNxrt[XYY2]MaxUseDefault	boolean
XtNxrt[XYY2]Min	single_float
XtNxrt[XYY2]MinUseDefault	boolean
XtNxrt[XYY2]Num	single_float
XtNxrt[XYY2]NumMethod	integer

UIL Argument Name	Argument Type
XtNxrt[XYY2]NumUseDefault	boolean
XtNxrt[XYY2]Origin	single_float
XtNxrtXOriginBase	single_float
XtNxrt[XYY2]OriginPlacement	integer
XtNxrt[XYY2]OriginUseDefault	boolean
XtNxrt[XYY2]Precision	integer
XtNxrt[XYY2]PrecisionUseDefault	boolean
XtNxrt[XYY2]Tick	single_float
XtNxrt[XYY2]TickUseDefault	boolean
XtNxrt[XYY2]Title	string
XtNxrt[XYY2]TitleRotation	integer
XtNxrt[YY2]AnnotationAngle	single_float
XtNxrt[YY2]AnnotationAngleUseDef [a]	boolean
XtNxrt[YY2]Axis100Percent	boolean
XtNxrtYAxisConst	single_float
XtNxrtYAxisMult	single_float

a. Argument name truncated to 31-character UIL limit

Reasons
XtNxrtExposeCallback
XtNxrtMapCallback
XtNxrtModifyCallback
XtNxrtPickCallback
XtNxrtResizeCallback
XtNxrtRotateCallback
XtNxrtTransformCallback
XtNxrtZoomAxisCallback

Index

_xrt.c 39
_xrt.h 40
_xrt.make 40
_xrt.res 40
_xrt_main.c 40
100% axis fill 72
2-byte fonts, use of 134
3D effect 70
  shading methods 71

A

action routines
  example of 118
  Map() 118, 252
  ModifyEnd() 116, 253
  ModifyStart() 111, 252
  NoAction() 116, 253
  Pick() 117, 253
  Properties() 254
  RotateUpdate() 115, 254
  ScaleUpdate() 112, 254
  TranslateUpdate() 112, 254
  ViewReset() 114, 254
  ZoomAxisCancel() 255
  ZoomAxisEnd() 113, 255
  ZoomAxisUpdate() 255
  ZoomCancel() 255
  ZoomEnd() 112, 255
  ZoomUpdate() 255
actions, listing of 251
adding a second Y-axis 127
adding scrollbars to graph 94
annotating axes 50
annotation
  formatting 66
  overview 50
  placing 65
  rotating 63
  setting font 64
  values
    formatting 66
annotation methods
  Point-labels 50
  Time-labels 50
  Value-labels 50
  values 50
ANSI C 39
ANSI C, compatibility with XRT/graph 11
arguments, UIL 283
ARRAY 215
ARRAY DATA 215
ARRAY data
  discussion of 13
Array data 99
ARRAY DATA with "T" 215
authorizing
  shared libraries 11
authorizing XRT/graph programs 10
axis
  annotation formatting 66
  annotation methods 50
  bounding box 66
  data styles 79
  datastyles 66
  font 64
  grid-lines 62
  inverting orientation 63
  labelling, overview 50
  logarithmic 52
  margins 64
  minimum and maximum 60
  numbering increment 51
  numbering method 50
  placing annotation and titles 65
  precision 51
  reversing direction of 64
  rotating annotation 63
  rotating titles 63
  tick increment 62
  title 63
axis bounds 60
Axis Data Styles 79
axis resources, summary 46
axis zooming
  default translation 251
AxisLabelFormat 66

B

background color
  Text Object 133
background colors 88
  data area 88
  footer 88
  graph 88

header 88
legend 88
window 88
background image 91
minimum image size 93
bar chart resources 46
bar charts
3D effect 70
3D shading method 71
cluster overlap 81
cluster width 80
special resources 80
stacking type 80
bar2tran.c example 18
bar2tran_cpp.cxx C++ example 18
border
footer 91
graph 91
header 91
legend 91
Text area 91
borders
Text Object 133
bubble charts
resources 84
bubble.c example 18
builder demo 18
introduction to 34

C

C++ 39
compatibility with XRT/graph 11
header file 140
use of 140
C++ classes
XrtGraph 140
XrtGraphData 140
XrtPointStyle 141
XrtTextArea 140
callback resources, summary 49
callback structures
Rotate 263
Transform 264
chart
background color 88
class hierarchy 171
class information 171
class name 171
class pointer 171
color field, in XrtDataStyle 75
Colors
Get Closest Color 30
colors
background 88
color database 29
flashing of colormap 29
foreground 88
monochrome use 29
specifying 29
strategies for use of 89
Text Object 133
combination graphs
graph types 128
legend 129
marker dataset 89
overview 128
second Data style 129
special rules 129
using Y2-axis 129
comments on product 4
common.c example 18
compiling generated code 41
compiling XRT/graph programs 10
controlling interactive rotate 115
controlling interactive scale, transform, zoom 113, 114
cross-platform applications with Microsoft Windows 136
customizing XRT/graph warnings 140

D

data
Array 99
attaching to graph 97
changing 105
default hole value 100
direct access 102
first point to graph 103
first set to graph 103
General 99
holes 100
last point to graph 104
last set to graph 103
multiple datasets 105
number of points 100
number of sets 100
real-time 97
real-time from pipe example 107
setting hole value 100
static 106
summary of procedures and methods 100
transposing 104
type 100
data area
background color 88
foreground color 89
data bounds 60
data from a file 97
data minimum and maximum 60
data object 98
display of 60
example of use 11
programming responsibilities 98
data resources, summary 43
Data style resources, summary 45
Data styles 74
accessing 78

axis 66, 79
combination graphs 129
default 74
fill patterns 76
grid 62
interactive editing 80
line patterns 76
marker 90
pie chart other slice 82
point styles 77
pointstyle 79
setting 77
setting for monochrome display 77
data types, listing of 257
dataset, definition of 13
default user interactions 109
demo programs, discussion of 18
destroy procedure 151
disabling all user interaction 112
disabling types of user interaction 116
double buffering graph updates 93
drill_down demo 18
dual Y-axis
on combination graphs 129
overview 127
relationship 127
dynamic and static colors, discussion 30

E

environment variables, XRTHOME 9
event handling procedures 119
example programs, listing of 18
Expose callback 112, 113, 115, 125, 174

F

Fast Update procedures 135
example 136
Expose callback triggering 135
use of 135
XrtArrCheckAxisBounds() 135
XrtArrDataFastUpdate() 135
XrtArrDataShiftPts() 135
XrtGenCheckAxisBounds() 135
XrtGenDataFastUpdate() 135
XrtGenDataShiftPts() 135
feedback demo 18
fill patterns 76
FirstPoint 103
FirstSet 103
flow demo 18
fonts
2-byte 134
setting, annotation/title 64
Text Object 132
fonts, specifying 73
footer
background color 88
border 91
example of use 13
font 73
foreground color 88
image 91
justification 72
positioning 70
text 72
footer area, definition 21
footer resources, summary 48
foreground color
Text Object 133
foreground colors 88
data area 89
footer 88
graph 88
header 88
legend 88
window 88
formatting axis annotation 66
fpat field, in XrtDataStyle 75
framing graph data 60
framing graph display 60

G

GENERAL 215
GENERAL DATA 216
GENERAL data
discussion of 13
General data 99
transposing 54
use of Point-labels 54
GENERAL DATA with “T” 216
generating source code 38
getting data into graphs 97
graph
background color 88
border 91
foreground color 88
image 91
margins 64
output 95
positioning 70
graph area positioning, resources 70
graph area, definition 21
graph areas
illustration 23
positioning 69
graph data 97
framing 60
graph data area
background color 88
foreground color 89
graph resources, summary 44
graph types 12
graph_editor demo 19

grid-lines
  Data style 62
  setting 62
GUI builders
  resource setting syntax 277

H

header
  background color 88
  border 91
  example of use 13
  font 73
  foreground color 88
  image 91
  justification 72
  positioning 70
  text 72
header area, definition 21
header resources, summary 48
hilo/candle chart resources, summary 47
hilow demo 18
holes in data 100
  default hole value 100

I

image 91
Image Functions
  Writing 147
Image Support
  Custom 145
  Predefined 144
include file, XRT/graph 171
instantiating graph widget 171
internationalization
  2-byte fonts 134
  ISO Latin 1 character set 134
  locale 133
  pie slice other label 82
  X fonts 134
introduction to XRT/graph 1
  summary of features 1
inverting graph orientation 63
  example 16
ISO Latin 1 character set 134

L

labelling axes, overview 50
labelling resources, summary 45
labels
  example of use 13
  other pie slice 82
  placing 65
  rotating 63
  setting font 64
LastPoint 104
LastSet 103
legend
  background color 88
  border 91
  combination graphs 129
  font 73
  foreground color 88
  HiLo, HiLoOpenClose, Candle and Bubble Chart 73
  image 91
  orientation 73
  positioning 70, 73
  text 73
legend area, definition 21
legend resources, summary 49
line patterns 76
linking
  shared libraries 11
locale
  pie slice other label 82
  XRT/graph support for 133
logarithmic axes 52
  rules for X-axis 52
lpat field, in XrtDataStyle 75

M

Map callback 118, 184
  example of use 118
Map() action 118, 184, 252
  use of XrtMap() 118
marker resources, summary 45
markers 89
  combination graphs 129
  data styles 90
  examples of use 90
  performance issues 90
  use in pie, polar, radar and filled radar charts 89
  X-marker positioning 90
  Y-marker positioning 90
missing data values 100
mktime() 57
Modify callback 112, 116, 184
ModifyEnd() action 116, 185, 253
ModifyStart() action 111, 185, 252
monochrome display
  colors 29
  Data styles setting caution 77
Motif integration 21
MtGraphTransposeData() 105
mvc demo 19

N

new features 2
NoAction() action 116, 253
NPoints 100
NSets 100

numbering axes, methods 50
numbering increment 51
numbering precision 51

O

orientation of graph 63
origin
  placing 61
  setting coordinates 61
other pie slice
  Data style 82
  disabling 82
  labelling 82
  threshold 82
outputting graphs 95

P

panel.c example 18
pcolor field, in XrtDataStyle 75
performance improvement
  resource updating 133
Pick callback 118, 185, 186
Pick() action 117, 185, 253
  use of XrtPick() 117
pie chart resources, summary 47
pie charts
  3D effect 70
  3D shading method 71
  differences from plot and bar charts 82
  disabling other slice 82
  exploding 83
  exploding, example 84
  minimum slices 83
  ordering of pies 83
  other slice Data style 82
  other slice label 82
  other slice threshold 82
  special resources 82
  starting angle for 84
  zero values 83
pie.c example 18
placing annotation and title 65
placing origins 61
plot1.c
  discussion of 11
  footer 13
  header 13
  labels 13
  listing of mm63.dat file 13
plot1.c example 18
point field, in XrtDataStyle 75
point styles 77
point, discussion of 13
Point-labels
  definition of 52
  use of 52
Pointstyle
  data styles 79
  objects summary 79
  XrtPointStyleCreate() 79
  XrtPointStyleDestroy() 79
  XrtPointStyleGetValues() 79
  XrtPointStyleSetValues() 79
polar demo 18
polar/radar chart resources, summary 47
porting to Microsoft Windows 136
positioning axis in graph 64
positioning graph areas 69
  strategies 70
positioning origins 61
PostScript output 95
  encapsulated PostScript 225
precision 51
  issues on dual-axis charts 128
  issues on dual-axis graphs 128
printing
  sample dialog box 96
printing graphs 95
product feedback 4
profile demo 19
Properties() action 254
property editor
  adding to your application 34
  enabling 34
  generating source code 38
  introduction 33
  source code 38
  using 35
psize field, in XrtDataStyle 75

R

radar.c example 18
Rasterfile output 95
real-time data 97
  pipe example 107
Registering Image Types 144
resetting interaction
  default translation 251
resetting interactions 114
Resize callback 125, 187
resizing windows 125
resource conversion to String 141
resource files
  comments in 277
  continuing lines 277
  example of 280
  resource syntax 277
  specifying Booleans 277
  specifying Data styles 278
  specifying enumerated values 277
  specifying graph data 278
  specifying strings 278
  specifying Value-labels 279
  use of 277

resource files, discussion 31
resource summary
  axis 46
  bar chart 46
  callback 49
  data 43
  Data style 45
  footer 48
  graph 44
  header 48
  hilo/candle chart 47
  labelling 45
  legend 49
  marker 45
  pie chart 47
  polar/radar chart 47
resources
  batching updates 133
  floating point 27
  inherited 203
  pointer 28
  string 28
  type and class listing 267
  UseDefault 27
reverse resource converters 141
reversing axis direction 64
Rotate callback 115, 187
Rotate callback structure 263
RotateUpdate() action 115, 187, 254
rotating annotation and title 63
rotation
  default translation 251
  real-time limitation 63
rotation, interactive 109
running generated code 41

S

sample code 18
  Data style editing dialog 80
  print dialog box 96
ScaleUpdate() action 112, 254
scaling
  default translation 251
scaling, interactive 109
scroll_time demo 18
scrollbars
  adding to graph 94
ScrolledWindow widget
  use with graph 94
scrollgraph.c example 18
second Y-axis 127
  relationship 127
set, discussion of 13
Set-labels
  definition of 52
  use of 52
shared libraries
  linking and authorizing 11
simple_uil.c example 18
slider.c example 18
source code
  generating from property editor 38
specifying fonts 73
Stacking 72
  100% axis fill 72
stacking bar charts 80
  100-percent format 81
stages of user interaction 111
static and dynamic colors, discussion 29
static data 106
stock.c example 18
stripper demo 19
Sun Rasterfile output 95

T

technical support 4
terminology of XRT/graph 23
Text areas
  batching creation/updates 133
  border 91
text areas 130
text object
  attachment methods 130
  attribute methods 205, 209
  attribute summary 131
  definition of 130
  instantiating 205, 209
  removing 133
text objects
  overview 130
text_area demo 19
tick increment 62
time.c example 18, 58
Time-axis
  criteria for use 56
  example of use 58
  format 57
  precision 58
  time base 56
  time units 56
Time-axis Labels 56
Time-axis, overview 56
Time-labels 56
  overview 56
titles
  placing 65
  rotating 63
  rotation along X-axis, limitation 63
titling an axis 63
Transform callback 113, 114, 190
  example of use 113
Transform callback structure 264
TranslateUpdate() action 112, 190, 254
translating
  default translation 251
translation, interactive 109

translations and actions 109
  disabling 112
  programming overview 109
  rotation 109
  scaling 109
  translation 109
  zoom 109
  see also user interaction
translations, listing of 251
transposing data 104
  MtGraphTransposeData() 105
Type 100

U

UIL
  example 141
  header file 141
  use of 141
UIL arguments 283
user interaction
  controlling rotate 115
  controlling scale, translate, zoom 113, 114
  customizing 109
  default translations and actions 251
  disabling all 112
  disabling particular interactions 116
  disabling types of interaction 116
  effect on Axis min/max resources 113, 114
  effect on Graph Margin resources 112
  ending 116
  event handling 119
  resetting 114
  resetting Margin resources 114
  rotation 109
  scaling 109
  starting 111
  translation 109
  updating 112
  with graph data 117
  zooming 109
user interaction, default 109
user interaction, programming
  three stages 111
using C++ with XRT/graph 140
Using Dynamic or Shared Libraries 148
using GUI Builders 277
using property editor with your application 34
using resource files 277
using the property editor 35
using UIL with XRT/graph 141

V

Value-labels
  clearing 54
  deleting 55
  getting 54
  setting 54
  uses for 55
Value-labels, definition 54
Values annotation, overview 50
ViewReset() action 114, 190, 254
vmgraph demo 19

W

warning messages
  customizing 140
widget
  class hierarchy 171
  class name 171
  class pointer 171
  functions/macros 171
  include file 171
  instantiating 171
  synopsis 171
width field, in XrtDataStyle 75
windows, resizing 125

X

XAllocColor() 30
XFreeColors() 30
XImage of graph 95
XmNscrollingPolicy 94
XmNworkWindow 94
XrmMergeDatabases() 140
XRT/graph
  basic terminology 23
  class hierarchy 171
  class information 171
  class name 171
  class pointer 171
  features summary 1
  functions/macros 171
  include file 171
  instantiating 171
  introduction to 1
  introduction to property editor 33
  new features 2
xrt_auth program
  flags 10
  use of 10
XRT_BORDER 172
XRT_BORDER_WIDTH 172
XRT_DATA_AREA_IMAGE 173
XRT_DATA_AREA_IMAGE_LAYOUT 173
XRT_DATA_GENERAL
  transposing data 54
  use of Point-labels 54
XRT_FOOTER_IMAGE 176
XRT_FOOTER_IMAGE_LAYOUT 176
XRT_GRAPH_IMAGE 178
XRT_GRAPH_IMAGE_LAYOUT 178
XRT_HEADER_IMAGE 181

XRT_HEADER_IMAGE_LAYOUT 181
XRT_IMAGE_LAYOUT 182
XRT_LEGEND_IMAGE_LAYOUT 183
XRT_TEXT_ADJUST 131, 205
XRT_TEXT_ANCHOR 132, 205
XRT_TEXT_ATTACH_DATASET 205
XRT_TEXT_ATTACH_PIXEL_X 205
XRT_TEXT_ATTACH_PIXEL_Y 205
XRT_TEXT_ATTACH_POINT 206
XRT_TEXT_ATTACH_SET 206
XRT_TEXT_ATTACH_TYPE 131, 206
XRT_TEXT_BACKGROUND 133
XRT_TEXT_BACKGROUND_COLOR 206
XRT_TEXT_BORDER 133, 206
XRT_TEXT_BORDER_WIDTH 133, 206
XRT_TEXT_FONT 132, 206
XRT_TEXT_FOREGROUND 133
XRT_TEXT_FOREGROUND_COLOR 206
XRT_TEXT_HEIGHT 208
XRT_TEXT_IMAGE 207
XRT_TEXT_IMAGE_LAYOUT 207
XRT_TEXT_IS_CONNECTED 132, 207
XRT_TEXT_IS_SHOWING 133, 207
XRT_TEXT_NAME 207
XRT_TEXT_OFFSET 132, 207
XRT_TEXT_PRINT_FONT 207
XRT_TEXT_ROTATION 133, 207
XRT_TEXT_STRINGS 131, 208
XRT_TEXT_WIDTH 208
XRT_TEXT_X 208
XRT_TEXT_Y 208
XRT_Y2ANNOTATION_ANGLE_USE_DEFAULT 201
XrtAdjust 257
XrtAlign 257
XrtAnchor 257
XrtAngleUnit 257
XrtAnnoMethod 258
XrtAnnoPlacement 258
XrtArgValToFloat() 27, 211
  use when porting to Windows 138
XrtArrCheckAxisBounds() 135, 211
XrtArrDataAppendPts() 212
XrtArrDataFastUpdate() 135, 212
XrtArrDataRemovePts() 213
XrtArrDataShiftPts() 135, 213
XrtAttachType 258
XrtAttachWindow() 137
XrtAxis 258
XrtBorder 258
XrtCallbackStruct structure 175, 187, 259
XrtCreate() 137
XrtCustomFormatValidate() 69, 213
XrtDataConcat() 213
XrtDataConvertToHandle() 214
XrtDataCopy() 214
XrtDataCreate() 214
XrtDataCreateFromFile() 215
XrtDataDestroy() 216
XrtDataExtractSet() 217
XrtDataGetDisplay() 104, 217
XrtDataGetFirstPoint() 103, 217
XrtDataGetFirstSet() 103, 217
XrtDataGetHole() 100, 217
XrtDataGetLastPoint() 104, 218
XrtDataGetLastSet() 103, 218
XrtDataGetNPoints() 100, 218
XrtDataGetNSets() 100, 218
XrtDataGetType() 100, 218
XrtDataGetXData() 102, 219
XrtDataGetXElement() 219
XrtDataGetYData() 102, 219
XrtDataGetYElement() 219
XrtDataHandle
  example of use 11
XrtDataHandle data object handle 98
XrtDataHandle object
  basic attributes 100
  using multiple 105
XrtDataRelease() 103, 220
XrtDataRemoveSet() 220
XrtDataSaveToFile() 220
XrtDataSetDisplay() 104, 220
  use with HiLo, HiLoOpenClose, candle, bubble charts 104
XrtDataSetFirstPoint() 103, 221
  reset by XrtDataSetNPoints() 222
XrtDataSetFirstSet() 103, 221
  reset by XrtDataSetNSets() 222
XrtDataSetHole() 221
XrtDataSetLastPoint() 104, 221
  reset by XrtDataSetNPoints() 222
XrtDataSetLastSet() 103, 222
  reset by XrtDataSetNSets() 222
XrtDataSetNPoints() 100, 222
XrtDataSetNSets() 100, 222
XrtDataSetType() 100, 222
XrtDataSetXData() 223
XrtDataSetXElement() 223
XrtDataSetYData() 223
XrtDataSetYElement() 224
XrtDataSort() 224
XrtDataStyle structure 75, 259
  color field 75
  fpat field 75
  lpat field 75
  pcolor field 75
  point field 75
  psize field 75
  width field 75
XrtDataTranspose() 104, 224
  use with pie charts 82
XrtDataUpdateDataValue() 224
XrtDeletePointLabel() 225
XrtDeletePointLabel2() 225
XrtDeleteSetLabel() 225
XrtDeleteSetLabel2() 225
XrtDestroyData() 225
XrtDisplay 259
XrtDrawPS() 95, 225

XRT_TEXT_PRINT_FONT attribute 207
XrtDrawPSColor 259
XrtDupDataStyles() 78, 228
use of with pointer resources 28
XrtDupStrings() 228
use of with pointer resources 28
XrtDupValueLabels() 228
XrtFillPattern 260
XrtFloatToArgVal() 27, 229
use when porting to Windows 138
XrtFreeDataStyles() 229
use of with pointer resources 28
XrtFreePropString() 229
XrtFreeStrings() 229
use of with pointer resources 28
XrtFreeTextHandles() 229
XrtFreeValueLabels() 229
XrtGenCheckAxisBounds() 135, 230
XrtGenDataAppendPt() 230
XrtGenDataFastUpdate() 135, 231
XrtGenDataRemovePt() 231
XrtGenDataShiftPts() 135, 231
XrtGenerateCode() 39
XrtGetImage() 95, 232
XrtGetNthDataStyle() 78, 232
XrtGetNthDataStyle2() 232
XrtGetNthFooterString() 232
XrtGetNthHeaderString() 233
XrtGetNthPointLabel() 54, 233
XrtGetNthPointLabel2() 54, 233
XrtGetNthSetLabel() 54, 233
XrtGetNthSetLabel2() 54, 233
XrtGetOtherSliceSets() 82, 234
XrtGetPropString() 138, 234, 246, 262
XrtGetTextHandles() 234
XrtGetValueLabel() 54, 234
XrtGetValues() 138, 235, 262
XRTHOME environment variable 9
XrtInsertNthDataStyle() 235
XrtInsertNthPointLabel() 235
XrtInsertNthPointLabel2() 235
XrtInsertNthSetLabel() 236
XrtInsertNthSetLabel2() 236
XrtLinePattern 260
XrtList
adding sorted items 150
introduction 149
sample code 149
XrtLoadGraphFromFile() 37
XrtMakeData() 236
XrtMakeDataFromFile() 13, 236
XrtMakeTime() 57, 237
XrtMap() 123, 237, 252, 263
example of use 124
XrtMapCallbackStruct structure 118, 184, 252, 260
XrtMapResult structure 123, 260
XrtMarkerMethod 261
XrtModifyCallbackStruct structure 112, 116, 185, 252, 253, 261
XrtMrmInitialize() 141, 237
XrtNumMethod 261
XrtOriginPlacement 261
XrtOutputPS() 95, 238
advantages of XrtDrawPS() 225
XrtOutputRaster() 95, 238
XrtOutputXwd() 95, 238
XrtPDSCmpDoubles() 154, 158
XrtPDSCmpFloats() 154, 159
XrtPDSCmpInts() 154, 159
XrtPDSCmpStrings() 154, 159
XrtPDSCmpStringsIgnoreCase() 154, 159
XrtPDSImage 91, 143
Creating 143
XrtPDSImage Methods 156
XrtPDSImageCopy() 156
XrtPDSImageCreate() 143, 156
XrtPDSImageCreateProc 147
XrtPDSImageDestroy() 156
XrtPDSImageFuncs 144
XrtPDSImageGetClipMask() 157
XrtPDSImageGetColorDepth() 157
XrtPDSImageGetFilename() 157
XrtPDSImageGetHeight() 157
XrtPDSImageGetImage() 158
XrtPDSImageGetWidth() 157
XrtPDSImageHeaderCompareProc 147
XrtPDSImageIsClipMastSet() 158
XrtPDSImageIsImage() 158
XrtPDSImageIsSet() 158
XrtPDSImageNameCompareProc 147
XrtPDSImageOutputProc 147
XrtPDSIsSorted() 162
XrtPDSList
adding items 149
checking for existence 151
comparing items 154
copying 149
creating 149
destroy procedure 151
destroying 149
destroying item memory 150
item count 153
moving items 155
removing items 150
retrieving entire array 153
retrieving item by index 152
searching 151
sorting 155
XrtPDSList Object 148
XrtPDSListAdd() 150, 160
XrtPDSListAppend() 150, 160
XrtPDSListCreate() 149, 150, 160
XrtPDSListDelete() 150, 160, 165
XrtPDSListDeleteAll() 161, 165
XrtPDSListDestroy() 149, 151, 161
XrtPDSListDestroyAll() 150, 151
XrtPDSListDestroyObject() 151, 161
XrtPDSListDestroyXtPointer() 151, 161
XrtPDSListDuplicate() 149, 161
XrtPDSListFind() 152, 154, 162

XrtPDSListGetCompareProc() 152, 160, 162, 164, 165, 166
XrtPDSListGetItem() 152, 162
XrtPDSListGetItemCount() 153, 162
XrtPDSListGetItems() 153, 162
XrtPDSListGetItemSize() 153, 160, 163
XrtPDSListGetUserData() 156, 163
XrtPDSListInsertAfter() 150, 163
XrtPDSListInsertBefore(150
XrtPDSListInsertBefore() 150, 163
XrtPDSListIsList() 151, 164
XrtPDSListIsSorted() 155, 164
XrtPDSListLookupItem() 152, 164
XrtPDSListMoveItemsAfter() 164
XrtPDSListMoveItemsBefore() 155, 165
XrtPDSListRemove() 150, 160, 165
XrtPDSListRemoveAll() 150, 161, 165
XrtPDSListSearchBackward() 153
XrtPDSListSearchForward() 153, 165
XrtPDSListSearchReverse() 166
XrtPDSListSetCompareProc() 152, 154, 155, 158, 159, 166
XrtPDSListSetDestroyProc() 151, 161, 167
XrtPDSListSetItem() 153, 167
XrtPDSListSetUserData() 156, 167
XrtPDSListSort() 155, 167
XrtPDSSetDestroyProc() 160, 161, 165
XrtPick() 120, 239, 253, 263
  example of use 122
  use with pie charts 122
XrtPickCallbackStruct structure 118, 186, 253, 261
XrtPickResult structure 120, 261
XrtPieOrder 262
XrtPieThresholdMethod 262
XrtPoint 262
XrtPointStyleHandle 79
XrtPopdownPropertyEditor() 37
XrtPopupPropertyEditor() 37
XrtProperty 262
  list of enumerations 267
XrtPropertyEditorEnable() 35, 241
XrtReason 262
XrtRegion 263
XrtRemoveNthDataStyle() 241
XrtRotate 263
XrtRotateCallbackStruct structure 115, 188, 254, 263
XrtSampleColorFunction() 30
XrtSaveDataToFile() 241
XrtSaveGraphToFile() 37
XrtSaveImageGeneric() 146
XrtSetNthDataStyle() 78, 243
XrtSetNthDataStyle2() 243
XrtSetNthFooterString() 243
XrtSetNthHeaderString() 244
XrtSetNthPointLabel() 54, 244
XrtSetNthPointLabel2() 54, 244
XrtSetNthSetLabel 54
XrtSetNthSetLabel() 244
XrtSetNthSetLabel2() 54, 244
XrtSetPropString() 138, 244, 246, 262
  resource syntax 277
XrtSetReverseConverters() 245
XrtSetValueLabel() 54, 245
XrtSetValues() 138, 245, 262
XrtShading 263
XrtTextAreaCreate() 130, 245
XrtTextAreaDestroy() 133, 246
XrtTextAreaGetValues() 131, 246
XrtTextAreaSetValues() 130, 247
XrtTextAttach() 247
XrtTextCreate() 247
XrtTextDestroy() 247
XrtTextDetach() 247
XrtTextDetail() 248
XrtTextUpdate() 248
XrtTimeUnit 263
XrtTransformCallbackStruct structure 113, 190, 254, 255, 264
XrtType 264
XrtUnmap() 124, 248
XrtUnpick() 122, 248
XrtUpdatePropertyEditor() 37
XrtValueLabel structure 54, 264
XrtZoomAxisCallbackStruct structure 114, 202, 254, 255, 265
Xt Error database
  customizing 140
  location 140
  modifying messages 140
XtAddCallback()
  example of use 113
XtAppWarningMsg() 140
XtConvertAndStore() 74
  use with XRT/graph reverse converters 245
XtCreateXrtGraph() 249
XtIsXrtGraph() 249
XtNxrt3dTransformCallback 113, 114
XtNxrtAngleUnit 87, 171
XtNxrtAxisBoundingBox 66, 172
XtNxrtAxisFont 64, 172
XtNxrtBackgroundColor 88, 172
XtNxrtBarClusterOverlap 81, 172
  limitation on stacking bar charts 81
XtNxrtBarClusterWidth 80, 172
XtNxrtBubbleMax 85
XtNxrtBubbleMin 85
XtNxrtCandleComplex 86, 173
XtNxrtColorFunc 30
XtNxrtData 98, 105, 173
XtNxrtData2 129, 172, 173
XtNxrtDataAreaBackgroundColor 88, 173
XtNxrtDataAreaForegroundColor 89, 173
XtNxrtDataStyles 78, 174
XtNxrtDataStyles2 129, 174
XtNxrtDataStyles2UseDefault 129, 174
XtNxrtDataStylesUseDefault 74, 174
XtNxrtDebug 174
XtNxrtDoubleBuffer 93, 174
XtNxrtExposeCallback 112, 113, 115, 125, 174, 254, 255

during Fast Update 135
XtNxrtFooterAdjust 72, 175
XtNxrtFooterBackgroundColor 175
XtNxrtFooterBorder 91, 175
XtNxrtFooterBorderWidth 91, 176
XtNxrtFooterFont 176
XtNxrtFooterForegroundColor 176
XtNxrtFooterHeight 176
XtNxrtFooterImageMinimumSize 93
XtNxrtFooterStrings 72, 176
XtNxrtFooterWidth 176
XtNxrtFooterX 176
XtNxrtFooterXUseDefault 176
XtNxrtFooterY 176
XtNxrtFooterYUseDefault 176
XtNxrtForegroundColor 88, 177
XtNxrtFrontDataset 129, 177
XtNxrtGraph3dShading 71, 177
XtNxrtGraphBackgroundColor 177
XtNxrtGraphBorder 91, 177
XtNxrtGraphBorderWidth 91, 177
XtNxrtGraphDepth 70, 115, 177
with interactive rotation 109
XtNxrtGraphForegroundColor 178
XtNxrtGraphHeight 179
XtNxrtGraphHeightUseDefault 179
XtNxrtGraphInclination 70, 115, 178, 188
XtNxrtGraphMarginBottom 64, 178, 190
user interaction 112
XtNxrtGraphMarginBottomUseDefault 178
XtNxrtGraphMarginLeft 64, 178, 190
user interaction 112
XtNxrtGraphMarginLeftUseDefault 178
XtNxrtGraphMarginRight 64, 179, 190
user interaction 112
XtNxrtGraphMarginRightUseDefault 179
XtNxrtGraphMarginTop 64, 179, 190
user interaction 112
XtNxrtGraphMarginTopUseDefault 179
XtNxrtGraphRotation 70, 115, 179, 188
XtNxrtGraphShowOutlines 179
XtNxrtGraphWidth 179
XtNxrtGraphWidthUseDefault 179
XtNxrtGraphX 180
XtNxrtGraphXUseDefault 180
XtNxrtGraphY 180
XtNxrtGraphYUseDefault 180
XtNxrtHeaderAdjust 72, 180
XtNxrtHeaderBackgroundColor 180
XtNxrtHeaderBorder 91, 180
XtNxrtHeaderBorderWidth 91, 180
XtNxrtHeaderFont 180
XtNxrtHeaderForegroundColor 180
XtNxrtHeaderHeight 181
XtNxrtHeaderImageMinimumSize 93
XtNxrtHeaderStrings 72, 181
XtNxrtHeaderWidth 181
XtNxrtHeaderX 181
XtNxrtHeaderXUseDefault 181
XtNxrtHeaderY 181
XtNxrtHeaderYUseDefault 181, 182
XtNxrtHiLoCloseShow 86, 181
XtNxrtHiLoOpenCloseFullWidth 86, 181
XtNxrtHiLoOpenShow 86, 181
XtNxrtInvertOrientation 63, 182
example 16
XtNxrtIsStacked 72
XtNxrtIsStacked2 72
XtNxrtLegendAnchor 73, 182
XtNxrtLegendBackgroundColor 182
XtNxrtLegendBorder 91, 182
XtNxrtLegendBorderWidth 91, 182
XtNxrtLegendFont 182
XtNxrtLegendForegroundColor 183
XtNxrtLegendHeight 183
XtNxrtLegendOrientation 73, 183
XtNxrtLegendReversed 73
XtNxrtLegendShow 183
XtNxrtLegendTitle 73
XtNxrtLegendWidth 183
XtNxrtLegendX 183
XtNxrtLegendXUseDefault 183
XtNxrtLegendY 183
XtNxrtLegendYUseDefault 183
XtNxrtMapCallback 118, 184, 252, 262
XtNxrtMarkerDataset 89, 129, 184
XtNxrtMarkerDataStyle 90, 184
XtNxrtMarkerDataStyleUseDefault 184
XtNxrtModifyCallback 112, 116, 184, 252, 253, 262
XtNxrtOtherDataStyle 82, 185
XtNxrtOtherDataStyleUseDefault 185
XtNxrtOtherLabel 82, 185
XtNxrtPickCallback 118, 185, 186, 253, 262
XtNxrtPieMinSlices 83, 186
XtNxrtPieOrder 83, 186
XtNxrtPieThresholdMethod 82
XtNxrtPieThresholdValue 82, 186
XtNxrtPointLabels 54, 187
use in combination graphs 129
XtNxrtPointLabels2 187
XtNxrtPolarAxisAllowNegatives 88
XtNxrtPolarHalfRange 88, 187
XtNxrtRepaint 133, 187
XtNxrtResizeCallback 125, 187
XtNxrtRotateCallback 115, 187, 254, 262
XtNxrtSetLabels 54, 188
XtNxrtSetLabels2 188
XtNxrtTextImageMinimumSize 93
XtNxrtTimeBase 56, 188
XtNxrtTimeFormat 56, 188
XtNxrtTimeFormatUseDefault 56, 189
XtNxrtTimeUnit 56, 189
XtNxrtTransformCallback 190, 254, 255, 262
XtNxrtType 12, 24, 80, 82, 190
XtNxrtType2 129, 190
XtNxrtXAnnoPlacement 65, 190
XtNxrtXAnnotationMethod 50, 191
Point-labels 52, 53
Time-labels 56

Value-labels 54
Values 50
XtNxrtXAnnotationRotation 63, 191
XtNxrtXAxisLabelFormat 192
XtNxrtXAxisLogarithmic 52, 192
XtNxrtXAxisMax 60, 193, 202
XtNxrtXAxisMaxUseDefault 193
XtNxrtXAxisMin 60, 193, 202
XtNxrtXAxisMinUseDefault 194
XtNxrtXAxisReversed 64, 194
XtNxrtXAxisShow 65, 194
XtNxrtXGrid 62, 194
XtNxrtXGridDataStyle 62, 195
XtNxrtXGridDataStyleUseDefault 62, 195
XtNxrtXGridUseDefault 62, 195
XtNxrtXLabels 54, 195
XtNxrtXMarker 90, 195
XtNxrtXMarkerDataStyle 90, 196
XtNxrtXMarkerDataStyleUseDefault 196
XtNxrtXMarkerMethod 90, 196
XtNxrtXMarkerPoint 90, 196
XtNxrtXMarkerSet 90, 196
XtNxrtXMarkerShow 90, 196
XtNxrtXMax 60, 197
XtNxrtXMaxUseDefault 197
XtNxrtXMin 60, 197
XtNxrtXMinUseDefault 197
XtNxrtXNum 51, 198
XtNxrtXNumMethod 50, 198
XtNxrtXNumUseDefault 198
XtNxrtXOrigin 61, 198
XtNxrtXOriginBase 87, 198
XtNxrtXOriginPlacement 61, 199
XtNxrtXOriginUseDefault 199
XtNxrtXPrecision 51, 199
XtNxrtXPrecisionUseDefault 51, 199
XtNxrtXTick 62, 200
XtNxrtXTickUseDefault 200
XtNxrtXTitle 63, 200
XtNxrtXTitleRotation 63, 200
XtNxrtY2AnnoPlacement 65, 190
XtNxrtY2AnnotationAngle 201
XtNxrtY2AnnotationAngleUseDefault 201
XtNxrtY2AnnotationMethod 50, 191
Value-labels 54
Values 50
XtNxrtY2AnnotationRotation 63, 191
XtNxrtY2Axis100Percent 81, 201
XtNxrtY2AxisLabelFormat 192
XtNxrtY2AxisLogarithmic 52, 192
relationship between Y-axes 128
XtNxrtY2AxisMax 60, 193, 202
XtNxrtY2AxisMaxUseDefault 193
XtNxrtY2AxisMin 60, 193, 202
XtNxrtY2AxisMinUseDefault 194
XtNxrtY2AxisReversed 64, 194
XtNxrtY2AxisShow 65, 194
XtNxrtY2Grid 62, 194
XtNxrtY2GridDataStyle 62, 195
XtNxrtY2GridDataStyleUseDefault 62, 195
XtNxrtY2GridUseDefault 62, 195
XtNxrtY2Labels 54, 195
XtNxrtY2Max 60, 197
XtNxrtY2MaxUseDefault 197
XtNxrtY2Min 60, 197
XtNxrtY2MinUseDefault 197
XtNxrtY2Num 51, 198
XtNxrtY2NumMethod 50, 198
XtNxrtY2NumUseDefault 198
XtNxrtY2Origin 61, 198
XtNxrtY2OriginPlacement 61, 199
XtNxrtY2OriginUseDefault 199
XtNxrtY2Precision 51, 128, 199
XtNxrtY2PrecisionUseDefault 51, 199
XtNxrtY2Tick 62, 200
XtNxrtY2TickUseDefault 200
XtNxrtY2Title 63, 200
XtNxrtY2TitleRotation 63, 200
XtNxrtYAnnoPlacement 65, 190
XtNxrtYAnnotationAngle 201
XtNxrtYAnnotationAngleUseDefault 201
XtNxrtYAnnotationMethod 50, 191, 207
Value-labels 54
Values 50
XtNxrtYAnnotationRotation 63, 191
XtNxrtYAxis100Percent 81, 201
XtNxrtYAxisConst 127, 201
XtNxrtYAxisLabelFormat 192
XtNxrtYAxisLogarithmic 52, 192
XtNxrtYAxisMax 60, 193, 202
XtNxrtYAxisMaxUseDefault 193
XtNxrtYAxisMin 60, 193, 202
XtNxrtYAxisMinUseDefault 194
XtNxrtYAxisMult 127, 201
XtNxrtYAxisReversed 64, 194
XtNxrtYAxisShow 65, 194
XtNxrtYGrid 62, 194
XtNxrtYGridDataStyle 62, 195
XtNxrtYGridDataStyleUseDefault 62, 195
XtNxrtYGridUseDefault 62, 195
XtNxrtYLabels 54, 195
XtNxrtYMarker 90, 195
XtNxrtYMarkerDataStyle 90, 196
XtNxrtYMarkerDataStyleUseDefault 196
XtNxrtYMarkerShow 90, 196
XtNxrtYMax 60, 197
XtNxrtYMaxUseDefault 197
XtNxrtYMin 60, 197
XtNxrtYMinUseDefault 197
XtNxrtYNum 51, 198
XtNxrtYNumMethod 50, 198
XtNxrtYNumUseDefault 198
XtNxrtYOrigin 61, 198
XtNxrtYOriginPlacement 61, 199
XtNxrtYOriginUseDefault 199
XtNxrtYPrecision 51, 199
XtNxrtYPrecisionUseDefault 51, 199
XtNxrtYTick 62, 200
XtNxrtYTickUseDefault 200
XtNxrtYTitle 63, 200

XtNxrtYTitleRotation 63, 200
XtNxrtZoomAxisCallback 201, 254, 255, 262
XtVaCreateManagedWidget(), example with
XRT/graph 11
XtXrt3d 22
XtXrtGraph 22, 171
xtXrtGraphWidgetClass 171
XWD output 95, 238
XtNxrt 66

Y

Y2-axis
relationship to Y-axis 127
removing 128
Y2-axis, use of 127

Z

ZoomAxisCancel() action 255
ZoomAxisEnd() action 113, 202, 255
ZoomAxisUpdate() action 255
ZoomCancel() action 255
ZoomEnd() action 112, 190, 255
zooming
default translation 251
zooming, interactive 109
ZoomUpdate() action 255