XRT/3d™ for Motif

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 3.0



Software Development Productivity

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

January 1999

Part No: KL3M30

Copyright © 1993-1999 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
Changes Introduced in this Release 2
New Features in XRT/3d 3.0 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/3d

1 Getting Started: Developing a Simple XRT/3d Program 9
1.1 Introduction 9
1.2 Environment Variables 9
1.3 Compiling and Authorizing a Program 10
1.4 The simple.c Program 11
1.5 The simple.c 3D Data 12
1.6 The simple.c Controls 13
1.7 Sample Code 13

2 XRT/3d Basics 15
2.1 Integration with Motif 15
2.2 Terminology 15
2.3 Floating-Point Resources 17
2.4 UseDefault Resources 18
2.5 Pointer Resources 18
2.6 Batching Resource Updates 19
2.7 Colors 19
2.8 Using Resource Files 21
2.9 Debugging 21

3 Programming XRT/3d: Common Functions 23
3.1 Resources 23
3.2 Axis Controls 29
3.3 Legend Display 29
3.4 Perspective 31
3.5 Axis Scaling 31
3.6 Axis Labelling 32
3.7 Grid Lines 33
3.8 Header and Footer Text 33
3.9 Header, Footer and Legend Fonts 34
3.10 Area Positioning 35
3.11 Area Borders 36
3.12 Foreground and Background Colors 36
3.13 Background Images 38
3.14 Double Buffering 40
3.15 Output and Printing 40

4 Programming XRT/3d: Surfaces and Bars 43
4.1 15 Basic Types of Surfaces and Bars 43
4.2 Bar Charts and Histograms 48
4.3 Contours and Zone Display 52
4.4 Mesh Controls 54
4.5 Surface Colors 55
4.6 Solid Surface 55

5 Programming XRT/3d: Scatter Plots 57
5.1 3 Basic Types of Scatter Plots 57
5.2 Controlling Symbol and Drop Line Style 58
5.3 Data Styles 58

6 XRT/3d Data 63
6.1 Data Overview 63
6.2 The Xrt3dData Structure 66
Regular Grid Data 66
Irregular Grid Data 67
Scatter Plot Data 68
Convenience Procedures 69

7 Programming User Interaction 71
7.1 Default User Interaction 71
7.2 Overview of Translations and Actions 73

7.3 Starting User Interaction . . . 73
7.4 Updating User Interaction . . . 74
Scaling, Translation, and Zooming . . . 74
Rotation . . . 76
Disabling and Disallowing Interactions . . . 77
7.5 Ending User Interaction . . . 78
7.6 Interacting with Graph Data . . . 78
Action Routines and Callbacks . . . 78
Event Handling Procedures . . . 80
7.7 Window Resizing . . . 84

8 Advanced XRT/3d Programming . . . 87

8.1 4D Surface Graphs . . . 87
8.2 4D Bar Charts . . . 88
8.3 Text Objects . . . 89
8.4 Customizing the Distribution Table . . . 93
8.5 Customizing Legend Labels . . . 94
8.6 Customizing Contour Styles . . . 95
8.7 Internationalization Support . . . 98
8.8 Porting XRT/3d to Microsoft Windows . . . 98
8.9 Using C++ with XRT/3d . . . 101
8.10 Using UIL with XRT/3d . . . 101
8.11 Converting Resource Values to Strings . . . 102

9 PDS Utility Library . . . 105

9.1 XrtPDSImage . . . 105
Creating an XrtPDSImage . . . 105
9.2 Registering Image Types . . . 106
Predefined Image Support . . . 106
Custom Image Support . . . 107
Writing the Image Functions . . . 109
9.3 Using Dynamic or Shared Libraries . . . 110
9.4 XrtPDSList Object . . . 110
Sample Code . . . 110
Creating and Destroying a List . . . 111
Adding and Deleting List Items . . . 111
Searching and Retrieving List Items . . . 113
List Comparison Routines . . . 115
Sorting a List . . . 116
User-Defined Data . . . 117

9.5 Procedures and Methods Reference 118
XrtPDSImage Methods 118
XrtPDSList Object Methods 120

Part II: Reference Appendices

A XRT/3d Resource Reference 133
A.1 Widget Synopsis 133
A.2 XRT/3d Resources 134
A.3 Inherited Resources 156

B XRT/3d Text Object Resource Reference 157
B.1 Object Synopsis 157
B.2 Text Object Resources 157
B.3 Inherited Resources 160

C Procedures and Methods Reference 161

D Translations and Actions Reference 187
D.1 Default Translations 187
D.2 Action Routines 188

E Data Types 193

F Resource Names, Types and Classes 205

G Resource File and GUI Builder Usage 217
G.1 Syntax 217
G.2 Example Resource File 220

H UIL Arguments 223

Preface

Introduction ■ *Changes Introduced in this Release*
New Features in XRT/3d 3.0 ■ *Assumptions*
Typographical Conventions Used in This Manual
Related Documents ■ *Technical Support*
Product Feedback and Announcements

Introduction

The XRT/3d Widget

Motif is composed of many different widgets such as buttons, sliders, and menus. Conceptually, XRT/3d adds a new widget to Motif. The widget displays 3D data in surface, contour, and bar chart representations.

XRT/3d has resources that determine how the graph will look and behave. Writing programs using XRT/3d is very similar to writing any other kind of Motif program; you now have one more Motif widget to use.

XRT/3d may be used in conjunction with KL Group's XRT/graph and XRT/table widgets. XRT/graph displays 2D data in X-Y plots, bar and pie charts. XRT/table displays rows and columns of tabular information in a scrollable window.

XRT/3d has resources which allow control of:

- Drawing (whether to draw the mesh, surface, contours and/or zones) and graph type (surface chart, bar chart or scatter plot).
- Contour styles: line widths, colors and patterns, and fill colors.
- Distribution method, distribution table, number of distribution levels.
- Hidden-line display, surface colors and mesh colors.
- 3D rotation and perspective.
- Legend positioning, orientation, style, border style, anchor, title font and color.
- Header and footer positioning, border style, text, font and color.
- Graph positioning, border style, color, width, and height.
- Axis maximums and minimums, fonts, gridlines, titles and labels.
- Window background and foreground color.

XRT/3d also provides several procedures and methods which:

- Allocate and load data structures containing the numbers to be graphed.
- Output a representation of the graph in PostScript or CGM format.
- Assist the developer in dealing with user-events.
- Assist the developer in dealing with interactive rotation, zooming and shifting of the 3D view by the end-user.
- Assist the developer with setting and getting indexed resources.

Changes Introduced in this Release

XRT/3d 3.0 is backwards-compatible with XRT/3d 2.1. If you have existing applications that you want to use with XRT/3d 3.0, you should be aware of the following changes in this release:

Recompiling Required	Existing applications only need to be recompiled to use this release of the library.

New Features in XRT/3d 3.0

Scatter Plots

A new chart type and a new data type have been added to produce Scatter Plot charts from three serial (x, y, z) data points. (See "3 Basic Types of Scatter Plots" on page 57.)

Grid Lines

A new data type, `Xrt3dLineStyle`, enables the definition of color, width and line styles for 3D chart grid lines. The default grid line style is a solid line of width 1 in the graph's foreground color (if specified), or the widget's foreground color. (See "Grid Lines" on page 33.)

Legend Control

A new resource allows the user to specify distribution intervals for Legend Labels. As well, a Legend Title resource is now available. (See "Legend Display" on page 29.)

Background Images

The widget, graph, header, footer, legend and text objects can now have background images. Images can be centered or tiled. (See "Background Images" on page 38.)

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 supported, and then XRT/3d need only interpret 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. (See "PDS Utility Library" on page 105.)

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/3d applications. An understanding of basic Motif concepts such as getting and setting resource values is required before continuing with this manual. See page page 3 for information on Motif documentation.

Typographical Conventions Used in This Manual

`Typewriter Font`	■ C language source code and examples of file contents. ■ XRT/3d 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*.
`...[XYZ]...`	Many XRT/3d resources used to specify axis information are similar for the X-, Y-, and Z-axis. The syntax `[XYZ]` used in an resource name refers to one of all of the X, Y, or Z versions of this resource. For example, "`XtNxrt3dXMin`, `XtNxrt3dYMin`, and `XtNxrt3dZMin`" is shortened to "`XtNxrt3d[XYZ]Min`".

Related Documents

Motif Programming Manual by Dan Heller, 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/3d 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/3d. You may purchase annual XRT/care agreements from your XRT Reseller or KL Group. XRT/care ensures you will receive all XRT/3d 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/3d from KL Group, contact KL Group for technical support; if you purchased XRT/3d 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/3d 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/3d 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/3d, 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/3d

1

Getting Started: Developing a Simple XRT/3d Program

Introduction ■ *Environment Variables*
Compiling and Authorizing a Program ■ *The simple.c Program*
The simple.c 3D Data ■ *The simple.c Controls*
Sample Code

1.1 Introduction

This chapter allows you to immediately try out XRT/3d by compiling and running an example program, introducing some fundamental XRT/3d programming concepts. The program, *simple.c*, graphs a basic mathematical function and allows a user to directly manipulate the four most important XRT/3d drawing controls: DrawMesh, DrawShaded, DrawContours, and DrawZones.

This program can be found online in *$XRTHOME/src/3d/examples/simple.c*.

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/3d distribution files.

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

```
echo $XRTHOME
```

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

Before attempting to use XRT/3d, 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/3d.

1.3 Compiling and Authorizing a Program

If the X, Motif and XRT/3d libraries and include files are installed in the standard places (normally */usr/lib* and */usr/include/X11* and */usr/include/Xm)*, *simple.c* may be compiled and authorized by entering:[1]

```
cc simple.c -I$(XRTHOME)/include \
    -L$(XRTHOME)/lib -o simple \
    -lxrt3d -lpdsutil -lXm -lXpm -lXt -lX11 -lm

$XRTHOME/bin/xrt_auth simple
```

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

xrt_auth

After the XRT/3d library has been linked into an executable application, it must be authorized with the *xrt_auth* utility. This utility will check that the machine you are building on is authorized to use XRT/3d, and will enable 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/3d.

If your application is not successfully enabled using *xrt_auth*, graphs 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*.

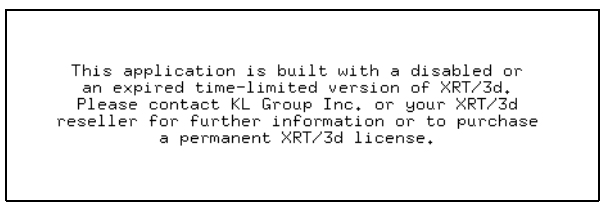


Figure 1 Message Displayed by an Expired or Non-authorized Application

1. Compilation options depend on your computer architecture. See the *Makefile* in *$XRTHOME/src/3d/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/3d *include* files are compatible with C++ and ANSI C as well as with traditional "Kernighan and Ritchie" C.

1.4 The simple.c Program

In this chapter we'll describe an introductory XRT/3d program called *simple.c*. This program, along with a Makefile, can be found in *$XRTHOME/src/3d/examples* .

simple.c is about 150 lines long. About 20% of the code deals with regular Motif widgets like the toggle widgets at the top of the form widget. This portion of the code should be easily understood by programmers already familiar with Motif. We'll concentrate on describing the XRT/3d portions of the code.

When *simple.c* is compiled and run, the window shown in Figure 2 is displayed. Try compiling and running *simple.c* before continuing with this section.

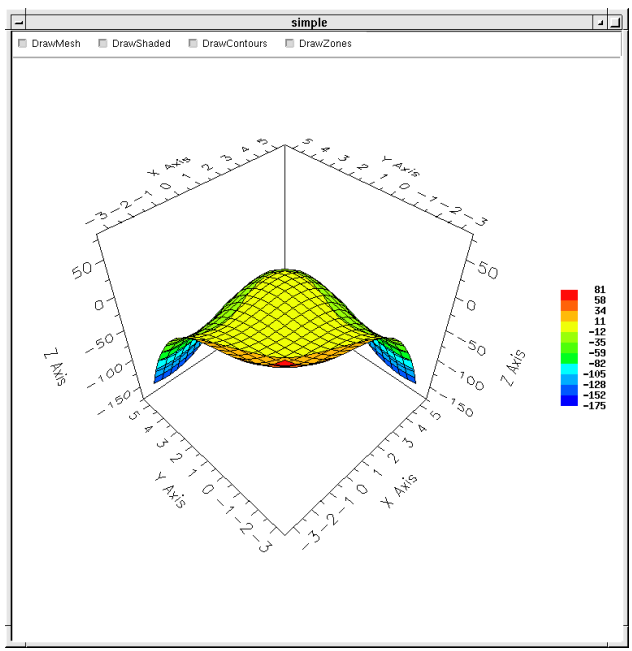


Figure 2 The Window Created by simple.c

simple.c illustrates several XRT/3d programming principles. Most of the initialization is handled by the *CreateWidgets()* routine. It creates a window containing a row of

check-box controls. Below the check-box controls, an XRT/3d widget is created. If the window is resized, the chart control is automatically resized.

1.5 The simple.c 3D Data

A common task in any XRT/3d program is to load the data to be graphed into an `Xrt3dData` structure. The data can come from any source: a file or database, a process running on another machine, or can be calculated on the fly. *simple.c* creates an `Xrt3dData` structure and loads it with grid data from the surface defined by the equation:

$$z = 3xy - x^3 - y^3 \quad -3 \le x, y \le 5$$

Chapter 6 on page 63 provides complete details on XRT/3d data, but for now, think of a grid aligned with the X- and Y-axis projected onto the surface we've defined above. When an X grid line crosses a Y grid line, the intersection defines a point on the surface. We'll evaluate the surface equation at that point, and pass the solution into the XRT/3d widget for graphing.

The first decision to make is: How many grid points? Too few grid points will result in too few samples, and a low resolution image. Too many will result in extra overhead, without adding clarity to the on-screen display.

simple.c defines the `Xrt3dData` structure in the *CalculateGrid()* routine. `numx` and `numy` are both assigned the value 20. They will define the grid resolution. *simple.c* calls *Xrt3dMakeGridData()* to allocate an `Xrt3dData` structure with enough space to hold 20 x 20 Z-values:

```
grid = Xrt3dMakeGridData(numx, numy,
    XRT3D_HUGE_VAL, 8.0/(numx-1), 8.0/(numy-1),
    -3.0, -3.0, True);
```

If *Xrt3dMakeGridData()* is successful, it returns a pointer to an `Xrt3dData` structure. If it is not successful, it returns `NULL`. The first two parameters, `numx` and `numy`, tell it how large a structure to allocate.

The third parameter indicates the data value that will be used to mean "no value". Since there are no holes in our surface data, we set this to `XRT3D_HUGE_VAL`, a value which is unlikely to appear in our data values.

The next two parameters specify the step distance between grid lines in the X and Y directions. Since our grid will have to span from -3 to 5 using 20 grid lines, the distance between grid lines will be 8.0 / (20 - 1) or about 0.421.

The next two parameters indicate the location of our data's origin (-3, -3). Finally, the last parameter tells *Xrt3dMakeGridData()* whether or not to allocate space for the Z values. We do not want to allocate this ourselves, so we set it to `True`.

Now that we have an `Xrt3dData` structure to hold our grid data, we can go ahead and evaluate the surface equation at each of the grid points. The *g.values* element in the

Xrt3dData structure is a two dimensional array of type double. The following code populates it with our surface's grid values:

```
for (i = 0; i < numx; i++) {
    x = grid->g.xorig + i*grid->g.xstep;
    for (j = 0; j < numy; j++) {
        y = grid->g.yorig + j*grid->g.ystep;
        grid->g.values[i][j] = 3*x*y - x*x*x - y*y*y;
    }
}
```

Our Xrt3dData structure referenced by grid is now ready for graphing. When *simple.c* creates the XRT/3d widget, it attaches the data to the widget by setting the XtNxrt3dSurfaceData resource to grid.

1.6 The simple.c Controls

At the top of the *simple.c* application window are a number of toggles that the user can adjust.

The DrawMesh, DrawShaded, DrawContours and DrawZones toggles turn the corresponding XtNxrt3dDrawMesh, XtNxrt3dDrawShaded, XtNxrt3dDrawContours and XtNxrt3dDrawZones Boolean resources on and off.

These four resources provide 15 different combinations of basic 3D data display. Some combinations will be more useful for your applications than others. These resources are described in section 3.2 on page 33.

Most XRT/3d resources are discussed in the following chapters, and all are fully described in Appendix A.

1.7 Sample Code

XRT/3d 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/3d/examples*:

- *simple.c*
- *bars.c*
- *texts.c*
- *scatter.c*
- *simple_cpp.cxx*
- *simple_uil.c*

Demos

The following more complex demos are located in *$XRTHOME/src/3d/demos*. Each demo has its own directory.

- *feedback3d* demonstrates use of the mapping and picking feedback functions. A user can pick grid points on a surface and drag points to new values.
- *math3d* graphs built-in mathematical functions. A user can enter a custom function to graph, as well as change many graph display options.
- *play3d* provides direct access to all XRT/3d and text object resources. This program can load and save graph descriptions in resource files, which you can use in your application.
- *shader* enhances the visual effect of surfaces by shading them. This program uses the *Xrt3dDataShaded()* procedure, which uses XRT/3d's 4D capabilities to shade each facet of a surface according to the angle and intensity of a light source.
- *spline* shows how to reduce the size of large datasets using cubic and linear spline interpolation. Reducing the size of the dataset improves XRT/3d's performance.
- *view_dir* shows how to display data as a 3D bar histogram. The program gives a visual display of files in a selected directory.

Multi-Product Demos

Several demos use XRT/3d with other XRT products. The following demos are located in their own directory in *$XRTHOME/src/common_demos*.

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

2

XRT/3d Basics

Integration with Motif ■ *Terminology*
Floating-Point Resources ■ *UseDefault Resources*
Pointer Resources ■ *Batching Resource Updates*
Colors ■ *Using Resource Files*
Debugging

2.1 Integration with Motif

Conceptually, XRT/3d 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/3d widget is subclassed from the XmPrimitive widget. This means that XmPrimitive and Core resources can be set in addition to the resources defined by the XRT/3d widget. For example, the `XmNborderWidth` resource (defined by XmPrimitive) can be used to adjust the width of the border surrounding the XRT/3d widget. Also, the `XtNwidth` and `XtNheight` resources (defined by Core) can be used to adjust the width and height of the XRT/3d widget.

2.2 Terminology

Figure 4 shows the major components of the XRT/3d widget. There are four major areas: Header, Footer, Legend and Graph. Each area has its own origin, width and height.

Unit Cube

The *unit cube* is defined to be the smallest cube which encloses the entire 3D scene (including the axis). Some XRT/3d resources, such as those that specify axis scaling and axis font sizes, have definitions that depend on the unit cube size.

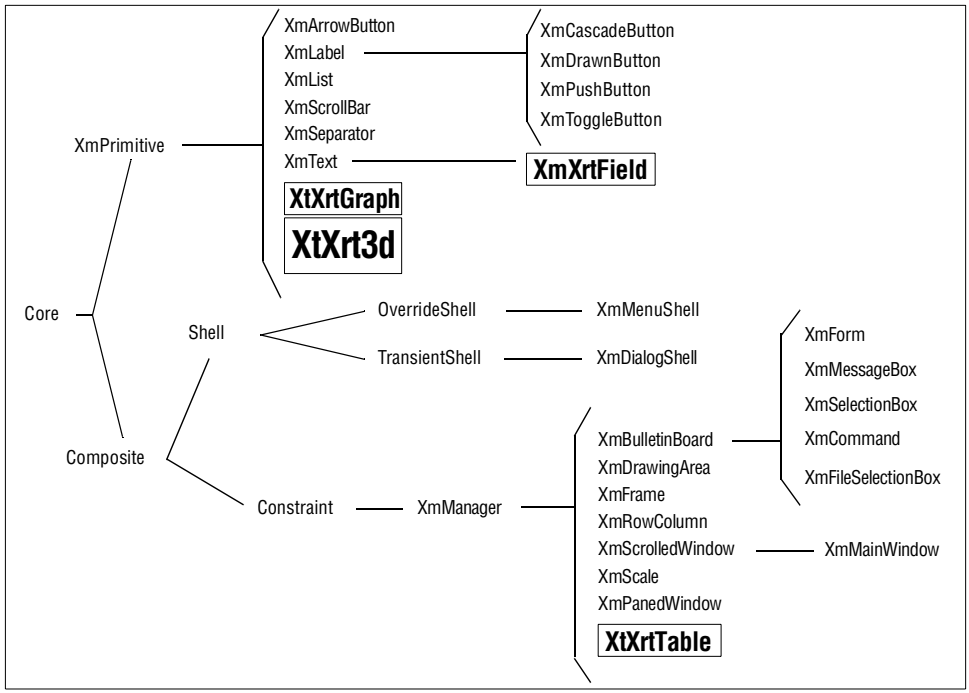


Figure 3 XRT/3d widget's position within the Motif Class Hierarchy

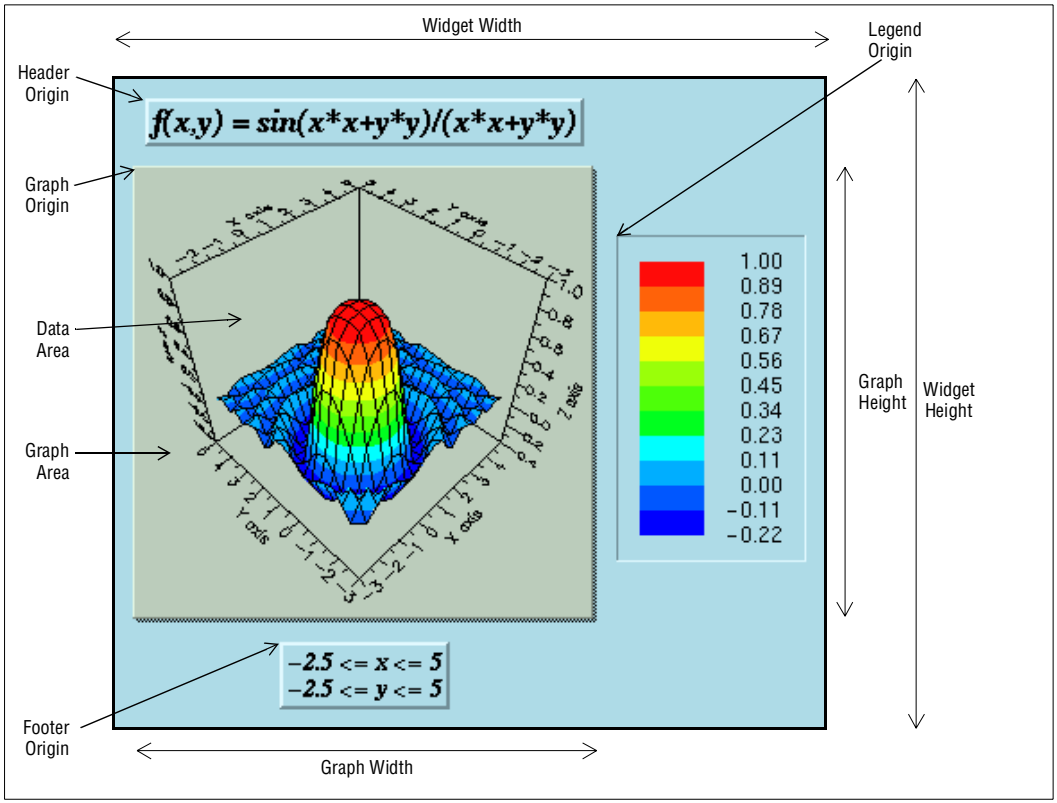


Figure 4 XRT/3d Areas

2.3 Floating-Point Resources

XRT/3d uses doubles for all floating-point resource values. When setting floating-point resources, the *Xrt3dDoubleToArgVal()* function must be used. For example, to triple the current Y-axis maximum:

```
double   ymax;
XtVaGetValues(mygraph, XtNxrt3dYMax, &ymax, NULL);
ymax = ymax * 3.0;
XtVaSetValues(mygraph,
    XtNxrt3dYMax, Xrt3dDoubleToArgVal(ymax),
    NULL);
```

2.4 UseDefault Resources

Many XRT/3d resources have corresponding `UseDefault` resources, for example, `XtNxrt3dHeaderX` and `XtNxrt3dHeaderXUseDefault`.

`UseDefault` resources are Booleans that determine if XRT/3d should calculate a default value for the resource.

For example, if `XtNxrt3dHeaderXUseDefault` is `True`, every time the XRT/3d window is resized, XRT/3d will determine a reasonable default value for the X pixel coordinate for the header area position. Otherwise, XRT/3d will use the last specified value of `XtNxrt3dHeaderX` to position the header area.

Attempts to set a `UseDefault` resource to `False` are ignored by XRT/3d unless the corresponding resource has been explicitly set or previously calculated by XRT/3d. 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 freeze the value of `XtNxrt3dXMax` at its current value. It will also have the side effect of setting `XtNxrt3dXMaxUseDefault` to `False`.

```
double  xmax;
XtVaGetValues(mygraph,
    XtNxrt3dXMax, &xmax, NULL);
XtVaSetValues(mygraph,
    XtNxrt3dXMax, Xrt3dDoubleToArgVal(xmax),
    NULL);
```

The following code will revert back to the default behavior, enabling XRT/3d to calculate a default value for `XtNxrt3dXMax` whenever it draws the graph.

```
XtVaSetValues(mygraph,
    XtNxrt3dXMaxUseDefault,  True,
    NULL);
```

Attempts to set `UseDefault` resources in resource files are ignored by XRT/3d.

2.5 Pointer Resources

Some XRT/3d 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".

The only exceptions to this rule are the `XtNxrt3dSurfaceData` and `XtNxrt3dZoneData` resources. These resources should always point to data structures that exist in program memory.

XRT/3d makes its own copy of all data passed through a pointer resource (except for `XtNxrt3dSurfaceData` and `XtNxrt3dZoneData`).

The "duplicating" procedures *Xrt3dDupContourStyles()*, *Xrt3dDupDistnTable()*, *Xrt3dDupDataStyles()*, *Xrt3dDupStrings()*, *Xrt3dDupValueLabels()*, and *Xrt3dDupXYColors()* can be used to make copies of some types of data. These copies

can be freed with corresponding “free” procedures (e.g. *Xrt3dFreeStrings()*). See Appendix C on page 161 for a complete list of procedures.

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 */
    XtNxrt3dHeaderStrings, &hs,
    XtNxrt3dFooterStrings, &fs
    NULL);

myhs = Xrt3dDupStrings(hs);     /* Make copies */
myfs = Xrt3dDupStrings(fs);

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

XtVaSetValues(graph,            /* Reset them */
    XtNxrt3dHeaderStrings, myhs,
    XtNxrt3dFooterStrings, myfs,
    NULL);

Xrt3dFreeStrings(myhs);         /* Free my copies */
Xrt3dFreeStrings(myfs);
```

2.6 Batching Resource Updates

Normally resource changes take effect immediately after the *XtSetValues()* or *XtVaSetValues* call. If you would prefer to make several changes to the widget’s resources before causing a repaint, set `XtNxrt3dRepaint` to `False`. All resource changes will be batched until `XtNxrt3dRepaint` is set to `True`.

2.7 Colors

You can specify colors for the foreground and background of the widget, as well as for each of the graph areas. The area backgrounds are transparent by default. The foreground colors default to the widget foreground color.

You can also specify colors for the top and bottom of the mesh, for surface shading, and for the contour lines and zone fills.

XRT/3d will choose default colors for data representation, so most 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, then 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 application built with XRT/3d is displayed on a monochrome X Server:

- XRT/3d will use ordered dithering to display zone colors and the surface top and bottom colors.
- All non-black colors appear as white when used for window or area background colors.
- All other non-white colors will appear as black.

Once a XRT/3d widget is created, an application should not update or change the application's colormap. Changes to any of the colors should be made by updating the appropriate widget resources (i.e. `XtNxrt3dBackgroundColor`, `XtNxrt3dContourStyles`, 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 minimize 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.

However, many applications require dynamic colors. In this case, each application will be given exactly the colors it asks for[2]. 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/3d applications will automatically allocate and de-allocate color cells in the Server as required by the application. This is done to minimize demands on the colormap resource by the

---

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.

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

XRT/3d application. If an XRT/3d application requests more colors than are available, black or white will be used instead of the unavailable color.

Colors may also flash when changing XRT/3d resources that affect colors. If a color is changed and XRT/3d recognizes that redrawing is unnecessary, it will only change the color lookup entry in the Server's colormap.

2.8 Using Resource Files

Sometimes it is better to use resource files instead of C code to set the value of resources. Resource 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/3d resources can be set in resource files. However, most resources are too complex or too detailed for end-users to want to modify them. Resource files for all of the widgets in an application can be combined to create an application defaults file.

For further information on resource files and their usage, please consult your Xt Intrinsics documentation. Appendix G on page 217 provides details on the syntax of resource settings when used in resource files, as well as a sample graph resource file.

2.9 Debugging

Setting `XtNxrt3dDebug` to `True` will cause XRT/3d to print messages to standard error regarding any inconsistencies it discovers (such as setting data's *xstep* or *ystep* to negative values).

3

Programming XRT/3d: Common Functions

Resources ■ *Axis Controls*
Legend Display ■ *Perspective*
Axis Scaling ■ *Axis Labelling*
Grid Lines ■ *Header and Footer Text*
Header, Footer and Legend Fonts ■ *Area Positioning*
Area Borders ■ *Foreground and Background Colors*
Background Images ■ *Double Buffering* ■ *Output and Printing*

3.1 Resources

This section summarizes all of the XRT/3d resources. It is not necessary to remember all the resources in order to program XRT/3d effectively. For most graphs, many resources may be left with their default settings.

This section will not describe each resource in detail. Each resource is summarized in Appendix A. Scan through these tables to gather a basic understanding of the resources. Not all the resources are used for all types of graphs: some are specific for surface and bar only, some for scatter plots only. See the following chapters for more details.

Resource	Type
XtNxrt3dContourStyles	Xrt3dContourStyle **
XtNxrt3dDataAreaBackgroundColor	char *
XtNxrt3dDataStyles	Xrt3dDataStyles **
XtNxrt3dDataStylesUseDefault	Boolean
XtNxrt3dDistnMethod	Xrt3dDistnMethod
XtNxrt3dDistnTable	Xrt3dDistnTable *
XtNxrt3dDrawDropLines	Boolean
XtNxrt3dDrawMesh	Boolean
XtNxrt3dDrawShaded	Boolean
XtNxrt3dDrawContours	Boolean
XtNxrt3dDrawZones	Boolean
XtNxrt3dDrawHiddenLines	Boolean
XtNxrt3dImage	Xtr3dImage
XtNxrt3dImageLayout	Xrt3dImageLayout
XtNxrt3dMeshBottomColor	char *
XtNxrt3dMeshTopColor	char *
XtNxrt3dNumDistnLevels	int
XtNxrt3dPerspectiveDepth	double
XtNxrt3dProjectZMax	int
XtNxrt3dProjectZMin	int
XtNxrt3dSolidSurface	Boolean
XtNxrt3dSurfaceBottomColor	char *
XtNxrt3dSurfaceTopColor	char *
XtNxrt3dSurfaceData	Xrt3dData *
XtNxrt3dTextList	WidgetList
XtNxrt3dType	Xrt3dType
XtNxrt3dViewNormalized	Boolean
XtNxrt3dViewScale	double
XtNxrt3dView[XY]Translate	double
XtNxrt3d[XY]MeshFilter	int
XtNxrt3d[XY]MeshShow	Boolean
XtNxrt3d[XYZ]Rotation	double
XtNxrt3d[XYZ]Scale	double
XtNxrt3dZoneData	Xrt3dData *
XtNxrt3dZoneMethod	Xrt3dZoneMethod

Figure 5 Data Display Resources

Resource	Type
XtNxrt3dGraphBackgroundColor	char *
XtNxrt3dGraphBorder	Xrt3dBorder
XtNxrt3dGraphBorderWidth	Dimension
XtNxrt3dGraphForegroundColor	char *
XtNxrt3dGraphHeight	Dimension
XtNxrt3dGraphHeightUseDefault	Boolean
XtNxrt3dGraphImage	Xrt3dImage
XtNxrt3dGraphImageLayout	Xrt3dImageLayout
XtNxrt3dGraphWidth	Dimension
XtNxrt3dGraphWidthUseDefault	Boolean
XtNxrt3dGraph[XY]	Position
XtNxrt3dGraph[XY]UseDefault	Boolean

Figure 6 Graph Resources

Resource	Type
XtNxrt3dAxisFont	XmFontList
XtNxrt3dAxisTitleFont	XmFontList
XtNxrt3dAxisStrokeFont	Xrt3dStrokeFont
XtNxrt3dAxisStrokeSize	Dimension
XtNxrt3dAxisTitleStrokeFont	Xrt3dStrokeFont
XtNxrt3dAxisTitleStrokeSize	Dimension
XtNxrt3d[XYZ]GridLineStyle	Xrt3dLineStyle *
XtNxrt3d[XYZ]GridLineStyleUseDefault	Boolean
XtNxrt3d[XY]DataLabels	char **
XtNxrt3d[XYZ]AnnoMethod	Xrt3dAnnoMethod
XtNxrt3d[XYZ]AxisShow	Boolean
XtNxrt3d[XYZ]AxisTitle	char *
XtNxrt3d[XYZ]GridLines	int
XtNxrt3d[XYZ]Max	double
XtNxrt3d[XYZ]MaxUseDefault	Boolean
XtNxrt3d[XYZ]Min	double
XtNxrt3d[XYZ]MinUseDefault	Boolean
XtNxrt3d[XYZ]ValueLabels	Xrt3dValueLabel **
XtNxrt3dZOrigin	double

Figure 7 Axis Resources

Resource	Type
XtNxrt3dHeaderAdjust	Xrt3dAdjust
XtNxrt3dHeaderBackgroundColor	char *
XtNxrt3dHeaderBorder	Xrt3dBorder
XtNxrt3dHeaderBorderWidth	Dimension
XtNxrt3dHeaderForegroundColor	char *
XtNxrt3dHeaderImage	Xrt3dImage
XtNxrt3dHeaderImageLayout	Xrt3dImageLayout
XtNxrt3dHeaderImageMinimumSize	Boolean
XtNxrt3dHeaderStrings	char **
XtNxrt3dHeaderFont	XmFontList
XtNxrt3dHeaderHeight	Dimension
XtNxrt3dHeaderWidth	Dimension
XtNxrt3dHeader[XY]	Position
XtNxrt3dHeader[XY]UseDefault	Boolean
XtNxrt3dFooterAdjust	Xrt3dAdjust
XtNxrt3dFooterBackgroundColor	char *
XtNxrt3dFooterBorder	Xrt3dBorder
XtNxrt3dFooterBorderWidth	Dimension
XtNxrt3dFooterFont	XmFontList
XtNxrt3dFooterForegroundColor	char *
XtNxrt3dFooterHeight	Dimension
XtNxrt3dFooterImage	Xrt3dImage
XtNxrt3dFooterImageLayout	Xrt3dImageLayout
XtNxrt3dFooterImageMinimumSize	Boolean
XtNxrt3dFooterStrings	char **
XtNxrt3dFooterWidth	Dimension
XtNxrt3dFooter[XY]	Position
XtNxrt3dFooter[XY]UseDefault	Boolean

Figure 8 Header and Footer Resources

Resource	Type
XtNxrt3d[XY]BarFormat	Xrt3dBarFormat
XtNxrt3d[XY]BarSpacing	double
XtNxrt3dXYColors	Xrt3dXYColor **

Figure 9 Bar Resources

Resource	Type
XtNxrt3dExposeCallback	XtCallbackList
XtNxrt3dMapCallback	XtCallbackList
XtNxrt3dModifyCallback	XtCallbackList
XtNxrt3dPickCallback	XtCallbackList
XtNxrt3dResizeCallback	XtCallbackList
XtNxrt3dRotateCallback	XtCallbackList
XtNxrt3dTransformCallback	XtCallbackList

Figure 10 Callback Resources

Resource	Type
XtNxrt3dLegendAnchor	Xrt3dAnchor
XtNxrt3dLegendBackgroundColor	char *
XtNxrt3dLegendBorder	Xrt3dBorder
XtNxrt3dLegendBorderWidth	Dimension
XtNxrt3dLegendDistnRange	Xrt3dDistnRange
XtNxrt3dLegendFont	XmFontList
XtNxrt3dLegendForegroundColor	char *
XtNxrt3dLegendHeight	Dimension
XtNxrt3dLegendImage	Xrt3dImage
XtNxrt3dLegendImageLayout	Xrt3dImageLayout
XtNxrt3dLegendLabelFilter	int
XtNxrt3dLegendLabelFunc	XtFunction
XtNxrt3dLegendOrientation	Xrt3dAlign
XtNxrt3dLegendShow	Boolean
XtNxrt3dLegendStrings	char **
XtNxrt3dLegendStyle	Xrt3dLegendStyle
XtNxrt3dLegendTitle	char *
XtNxrt3dLegendWidth	Dimension
XtNxrt3dLegend[XY]	Position
XtNxrt3dLegend[XY]UseDefault	Boolean

Figure 11 Legend Resources

Resource	Type
XtNxrt3dBackgroundColor	char *
XtNxrt3dColorFunc	XtFunction
XtNxrt3dDebug	Boolean
XtNxrt3dDoubleBuffer	Boolean
XtNxrt3dForegroundColor	char *
XtNxrt3dPreviewMethod	Xrt3dPreviewMethod
XtNxrt3dRepaint	Boolean

Figure 12 Other Resources

Resource	Type
XtNxrt3dTextAdjust	Xrt3dAdjust
XtNxrt3dTextAttachIndex[XY]	int
XtNxrt3dTextAttachIndexPoint	int
XtNxrt3dTextAttachIndexSeries	int
XtNxrt3dTextAttachMethod	Xrt3dTextAttachMethod
XtNxrt3dTextAttachPixel[XY]	Position
XtNxrt3dTextAttachPoint[XYZ]	double
XtNxrt3dTextBackgroundColor	char *
XtNxrt3dTextBorder	Xrt3dBorder
XtNxrt3dTextBorderWidth	Dimension
XtNxrt3dTextFont	XmFontList
XtNxrt3dTextForegroundColor	char *
XtNxrt3dTextImage	Xrt3dImage
XtNxrt3dTextImageLayout	Xrt3dImageLayout
XtNxrt3dTextImageMinimumSize	Boolean
XtNxrt3dTextLineShow	Boolean
XtNxrt3dTextOffset[XY]	Position
XtNxrt3dTextPlane	int
XtNxrt3dTextPrintFont	char *
XtNxrt3dTextShow	Boolean
XtNxrt3dTextStrings	char **
XtNxrt3dTextStrokeFont	Xrt3dStrokeFont
XtNxrt3dTextStrokeSize	Dimension

Figure 13 Text Object Resources

3.2 Axis Controls

The resources listed in Figure 7 on page 25 control the Axis display.

Axis Show

The XtNxrt3d[XYZ]AxisShow resources tell XRT/3d whether it should draw the axis at all. If set to False, the axis will not be drawn.

Axis Font and Size

The axis annotation is normally rendered using the stroke font specified by XtNxrt3dAxisStrokeFont. There are several stroke fonts to choose from (see Appendix E). The default font is XRT3D_SF_ROMAN_SIMPLEX.

The axis font size is measured in units which are each 1/1,000 of the unit cube length. The default axis font size is 80, which means the characters are 8% of the length of the unit cube high. See section 2.2 on page 15 for a definition of the unit cube.

Title

The XtNxrt3d[XYZ]Title resources may be used to specify a title for each axis. Titles are rendered in the stroke font specified by XtNxrt3d[XYZ]AxisTitleStrokeFont, and in the size specified by XtNxrt3d[XYZ]AxisTitleStrokeSize.

Min and Max

The XtNxrt3d[XYZ]Max and XtNxrt3d[XYZ]Min resources specify the minimum and maximum values of each axis. By default, XRT/3d determines the extent of the axes based on the minimum and maximum data values and anytime the data changes, the axes update automatically.

You can frame the data displayed by setting these resources. When set by an application, the axis extents do not change when the data changes. To return to the default behavior, set XtNxrt3d[XYZ]MaxUseDefault or XtNxrt3d[XYZ]MinUseDefault to True.

Note: The Z-axis minimum/maximum cannot be inside the Z-range of the data. When XtNxrt3dZMin/XtNxrt3dZMax has been set, any changes to the data that put it inside the Z-range causes XRT/3d to set its corresponding UseDefault resource to True.

3.3 Legend Display

XRT/3d will automatically generate a legend whenever contours, zones or points are drawn. If you don't want a legend, set XtNxrt3dLegendShow to False. If contours are drawn and zones are not, a legend listing the contour lines is generated. If zones are drawn, the legend lists the fill colors for each level. If points are drawn, the legend lists the symbols used for each series of points.

By default, XRT/3d will attempt to list the legend contents vertically and position the legend to the right (i.e. east) of the graph area.

Orientation

The legend layout is controlled through the `XtNxrt3dLegendOrientation` resource. It may be either `XRT3D_ALIGN_VERTICAL` or `XRT3D_ALIGN_HORIZONTAL`. If the legend is too large to fit in one row or column, XRT/3d will create the legend in several rows or columns.

Anchor

The default legend positioning relative to the graph area is controlled with the `XtNxrt3dLegendAnchor` resource. Valid values correspond to the eight points of the compass: `XRT3D_ANCHOR_NORTH`, `XRT3D_ANCHOR_SOUTH`, `XRT3D_ANCHOR_EAST`, `XRT3D_ANCHOR_WEST`, `XRT3D_ANCHOR_NORTHWEST`, `XRT3D_ANCHOR_NORTHEAST`, `XRT3D_ANCHOR_SOUTHEAST`, and `XRT3D_ANCHOR_SOUTHWEST`.

Style

The `XtNxrt3dLegendStyle` resource can be set to either `XRT3D_LEGEND_STYLE_CONTINUOUS` (the default) or `XRT3D_LEGEND_STYLE_STEPPED`. See Figure 14 for an example of these styles. Continuous legends are recommended since they display the same information while using less space.

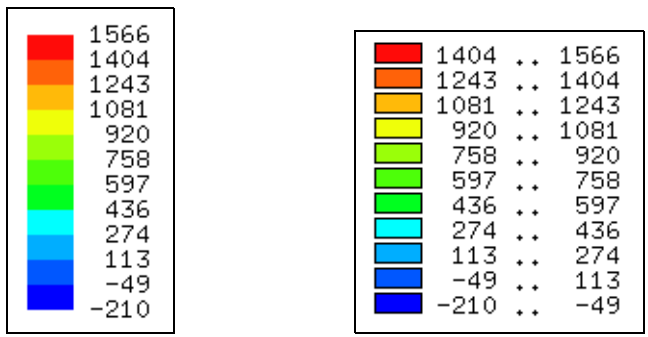


Figure 14 Continuous (left) and Stepped (right) Legends

Legend Label Options

The number of legend labels can be reduced using `XtNxrt3dLegendLabelFilter`. This can also be used to adjust stepped legends to take into account missing ranges. The resource is set to *n*, resulting in every *n*th legend glyph/label combination being drawn.

Legend Position and Border

To explicitly position the legend, see section 3.10 on page 35. Also, see section 3.11 on page 36 for information on how to program the legend border.

Legend Title

The legend title is a single line of text, centered over the top of the legend. It is specified in `XtNxrt3dLegendTitle`.

Custom Legend Contents

It is possible to specify custom text for the legend elements. See section 8.5 on page 94 for more information.

3.4 Perspective

The `XtNxrt3dPerspectiveDepth` resource controls the perspective effect observed by projecting the unit cube onto the screen. Small values exaggerate the perspective effect, while large values diminish it. Valid values are anything between 1 and `XRT3D_HUGE_VAL`. Although perspective can be set for scatter plots, it has no real meaning.

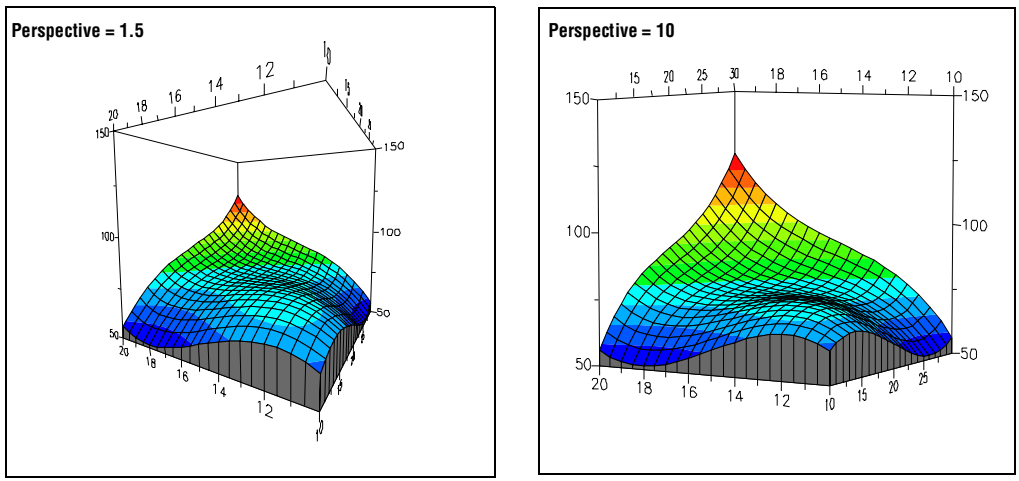


Figure 15 Perspective Depth Measurement

3.5 Axis Scaling

The axis scaling resources `XtNxrt3d[XYZ]Scale` can be used to adjust the X, Y and Z dimensions of the unit cube relative to one another. For example, if you would like the 3D display to be twice as long in the Y direction as in the X, set the Y scale to twice the X scale. Setting the Z scale higher or lower has the effect of flattening or stretching the surface view.

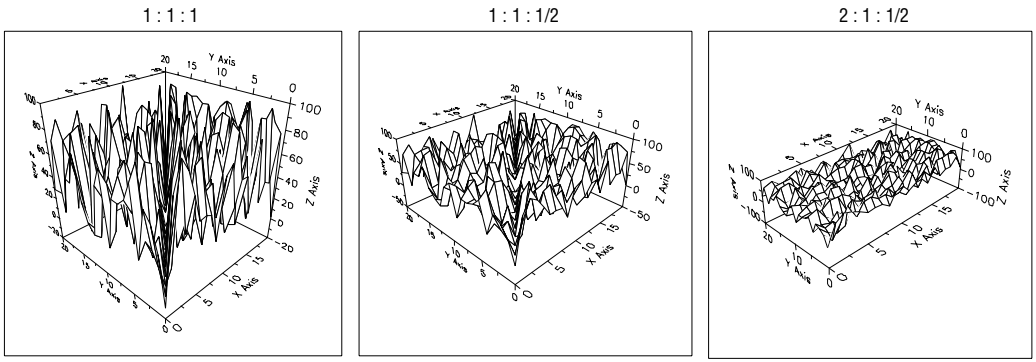


Figure 16 Axis scaling using various X : Y : Z ratios

3.6 Axis Labelling

There are three distinct ways to annotate the X- and Y-axes, and two ways the Z-axis can be annotated. The annotation method is determined by the `XtNxrt3d[XYZ]AnnoMethod` resource which takes one of the following three values:

```
typedef enum {
    XRT3D_ANNO_VALUE,
    XRT3D_ANNO_DATA_LABELS,     /* X, Y axes only */
    XRT3D_ANNO_VALUE_LABELS
} Xrt3dAnnoMethod;
```

Value Method

When `XtNxrt3d[XYZ]AnnoMethod` is `XRT3D_ANNO_VALUE`, XRT/3d will automatically annotate the axis based on the range of data. It is most suitable for the Z-axis, and for the X- and Y-axes when the value of `XtNxrt3dType` is `XRT3D_TYPE_SURFACE`. This is the default annotation method.

Data Labels Method

To label individual lines in a surface, or a row/column of bars, use `XRT3D_ANNO_DATA_LABELS`. This method uses a list of strings supplied with the `XtNxrt3d[XY]DataLabels` resource to annotate each line from the grid. Individual labels can be supplied with the procedure *Xrt3dSetNthDataLabel()* and retrieved with *Xrt3dGetNthDataLabel()* procedure. This labelling method will not display any labels when used with point data.

Value Labels Method

Labels can be placed at explicit locations along an axis by setting the value of `XtNxrt3d[XYZ]AnnoMethod` to `XRT3D_ANNO_VALUE_LABELS`. The locations of the labels

are specified with the resource XtNxrt3d[XYZ]ValueLabels, which takes an array of pointers to the following structure:

```
typedef struct {
    double      value;
    char *      label;
} Xrt3dValueLabel;
```

Any labels which are beyond the bounds of the axis are not drawn. Individual value labels can be added and removed through the procedure *Xrt3dSetValueLabel()*, and retrieved through the procedure *Xrt3dGetValueLabel()*.

3.7 Grid Lines

Grid lines can be displayed on each of the three primary planes using the XtNxrt3d[XYZ]GridLines resources. These resources take integers as values, combined from the following constants:

```
#define XRT3D_XY_PLANE      1
#define XRT3D_XZ_PLANE      2
#define XRT3D_YZ_PLANE      4
```

For instance, to draw X grid lines in all applicable planes, set XtNxrt3dXGridLines to the value (XRT3D_XY_PLANE | XRT3D_XZ_PLANE). To remove grid lines completely, set the value to 0. In a 2D graph, grid lines are displayed whenever the XtNxrt3d[XY]GridLines resource is any of the above values.

Grid lines are drawn where annotation is drawn on the axis, regardless of the annotation method.

Grid line color, width and line style are controlled by the XtNxrt3d[XYZ]GridLineStyle resources. Default grid line style is a solid 1-pixel wide line, using XtNxrt3dGraphForegroundColor or, if that is NULL, XtNxrt3dForegroundColor. To return to the default setting, use XtNxrt3d[XYZ]GridLineStyleUseDefault.

3.8 Header and Footer Text

The header and footer areas can both contain multiple lines of text. Text is aligned within the area, depending on the value of the XtNxrt3dHeaderAdjust or XtNxrt3dFooterAdjust resource. The values XRT3D_ADJUST_LEFT, XRT3D_ADJUST_RIGHT and XRT3D_ADJUST_CENTER cause the text to be left-justified, right-justified, or centered. XRT3D_ADJUST_CENTER is the default.

Text for the header and footer areas is specified using the XtNxrt3dHeaderStrings and XtNxrt3dFooterStrings resources. Both of these resources have NULL-terminated arrays of strings as their values.

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

```
static char *hs[] = { "Analysis Results",
                      "Sample Group A", NULL };
XtVaSetValues(my_graph,
    XtNxrt3dHeaderStrings,  hs,
    XtNxrt3dHeaderAdjust,   XRT3D_ADJUST_LEFT,
    NULL);
```

3.9 Header, Footer and Legend Fonts

Any X Window font may be specified for each of the header, footer and legend areas.

If you choose to use a named font, use the X Window program *xlsfonts* or the Xlib procedure *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.

```
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,
                XtNxrt3dHeaderFont,  hf,
                XtNxrt3dFooterFont,  ff,
                NULL);
}
```

Figure 17 Setting XmFontList resources

3.10 Area Positioning

At run-time, each of the four areas (header, footer, legend and graph) will, by default, be positioned by XRT/3d. 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 `XtNxrt3dLegendAnchor` resource.
- The positioning of areas which have been explicitly positioned by the user program.

XRT/3d'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.

`XtNxrt3dHeaderX`	`XtNxrt3dHeaderY`	`(XtNxrt3dHeaderWidth)`	`(XtNxrt3dHeaderHeight)`
`XtNxrt3dFooterX`	`XtNxrt3dFooterY`	`(XtNxrt3dFooterWidth)`	`(XtNxrt3dFooterHeight)`
`XtNxrt3dLegendX`	`XtNxrt3dLegendY`	`(XtNxrt3dLegendWidth)`	`(XtNxrt3dLegendHeight)`
`XtNxrt3dGraphX`	`XtNxrt3dGraphY`	`XtNxrt3dGraphWidth`	`XtNxrt3dGraphHeight`

(Resources in Parentheses are Read-Only)

Figure 18 Area Positioning Resources

When used in an *XtGetValues()* call, positioning resources will return the values used the last time the widget was displayed. The width and height resources are of type `Dimension`, and the X and Y resources are of type `Position`. `Dimension` and `Position` are defined by the Xt Intrinsics.

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 7.7 on page 84 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/3d calculate default positioning for all areas when the widget 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/3d.

3.11 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 BorderWidth resource for each of these areas (for example, `XtNxrt3dGraphBorder` and `XtNxrt3dGraphBorderWidth`).

The Border resources may be set to any of: `XRT3D_BORDER_NONE`, `XRT3D_BORDER_PLAIN`, `XRT3D_BORDER_3D_OUT`, `XRT3D_BORDER_3D_IN`, `XRT3D_BORDER_ETCHED_IN`, `XRT3D_BORDER_ETCHED_OUT`, `XRT3D_BORDER_FRAME_IN`, `XRT3D_BORDER_FRAME_OUT`, `XRT3D_BORDER_BEVEL` or `XRT3D_BORDER_SHADOW`. The width must be between 0 and 20.

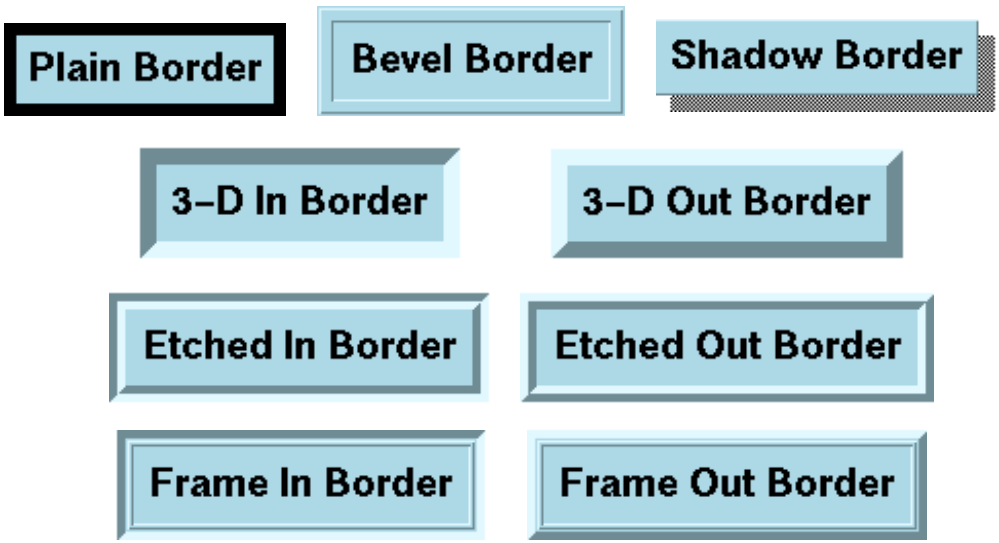


Figure 19 Border Types

If widget areas are explicitly positioned to intersect each other, highlighting the intersections by showing borders may look unattractive.

3.12 Foreground and Background Colors

Background Colors

The window background color is specified through the `XtNxrt3dBackgroundColor` 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 `XtNxrt3dLegendBackgroundColor`. The data area of the graph also has its own background color, specified by `XtNxrt3dDataAreaBackgroundColor`.

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 `XtNxrt3dForegroundColor` 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 `XtNxrt3dHeaderForegroundColor` resource.

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

Get Closest Color

XRT/3d 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/3d allocates either black or white, depending on the color the user is trying to allocate. The `XtNxrt3dColorFunc` 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/3d/util/x_color_func.c*. It is also included in the XRT/3d library as the method *Xrt3dSampleColorFunction()* .

The `XtNxrt3dColorFunc` 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 function you supply must have the following semantics to work with XRT/3d:

1. It must have the function prototype of the function below.
2. It must allocate a color (using *XAllocColor()*) because XRT/3d 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/3d 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/3d calls this function with color = `NULL` and do-free = `True` when the widget is destroyed).

Color function prototype:

```
unsigned long (*xrt3d_color_func)(
    Display     *display,
    int         screen,
    Colormap    colormap,
    XColor      *color,
    Boolean     do-free
    )
```

Specifying F/B Colors

All foreground and background resources take a color string (as described in section 2.7 on page 19) 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,
    XtNxrt3dBackgroundColor, "LightBlue",
    XtNxrt3dForegroundColor, "NavyBlue",
    XtNxrt3dHeaderBackgroundColor, "LightGray",
    XtNxrt3dFooterForegroundColor, "DarkGreen",
    NULL);
```

Take care not to choose a background color that is also used to display data in the graph.

3.13 Background Images

Each of the 4 graph areas (header, footer, legend and graph) may be enhanced with a background image. There is an Image and an Image Layout resource for each of these areas (for example, `XtNxrt3dGraphImage` and `XtNxrt3dGraphImageLayout`). In addition, you can specify an image for the entire widget.

The Image resources are of type `Xrt3dImage`. The `Xrt3dImage` type is actually the same as the `XrtPDSImage` type discussed in section 9.1 on page 105.

The following example shows how you can use XRT/3d's converters to set a background image in the footer:

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

void
set_image(Widget graph, char *file_name)
{
    Xrt3dImage  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,
        XtRXrt3dImage, &to)) {
        fprintf(stderr, "Conversion of %s to XtRXrt3dImage failed\n",
                file_name);
        image = NULL;
    }
    XtVaSetValues(graph,
        XtNxrt3dFooterImage,    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 *Xrt3dSetPropString()*.

```
    Xrt3dSetPropString(graph, XRT3D_FOOTER_IMAGE,
                    "$XRTHOME/src/icons/shading.xpm");
```

This function uses XRT/3d'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/3d to support custom image types, see section 9.1 on page 105.

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

- `XRT3D_IMAGELAYOUT_CENTERED` places the image as-is in the center of the area.
- `XRT3D_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.

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 `XtNxrt3dTextImageMinimumSize`, `XtNxrt3dHeaderImageMinimumSize` or `XtNxrt3dFooterImageMinimumSize`. 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 `XtNxrt3dTextImageMinimumSize`.

3.14 Double Buffering

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

When `XtNxrt3dDoubleBuffer` is `True`, every time XRT/3d changes a graph, it will:

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

When `XtNxrt3dDoubleBuffer` is `False`, every time XRT/3d 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, `XtNxrt3dDoubleBuffer` is `True`. However, setting it to `False` can improve the graphing performance and reduce memory requirements on the Server.

3.15 Output and Printing

Many applications need to provide users with a way to get a hardcopy of a graph they see on the screen. XRT/3d provides several procedures that output graph representations to files or printers. These procedures work correctly even when the XRT/3d widget is obscured by other windows. These procedures are fully described in Appendix C.

Xrt3dDrawPS()

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 image sizing/positioning. See page page 166 in Appendix C for complete details on using this procedure.

Xrt3dOutputPS()

Like *Xrt3dDrawPS()*, *Xrt3dOutputPS()* also creates an encapsulated PostScript description of the XRT/3d widget image, but 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. See page page 177 in Appendix C for complete details on using this procedure.

Xrt3dDrawCGM()

Xrt3dDrawCGM() creates a Computer Graphics Metafile (CGM) format description of the graph image. See page page 165 in Appendix C for complete details on using this procedure.

Xrt3dOutputXwd()

Xrt3dOutputXwd() creates an X Window Dump format description of the graph image. See page page 178 in Appendix C for complete details on using this procedure.

Xrt3dGetImage()

Xrt3dGetImage() returns a pointer to an *XImage* representation of the graph image. This may be used by developers who wish to implement their own unique output formats. See page page 171 in Appendix C for complete details on using this procedure.

Other Image Formats

XRT/3d can also output a graph in image formats that have been preregistered or registered by the user (See “Registering Image Types” on page 106.). The functions *Xrt3dSaveImageGeneric()*, *Xrt3dSaveImageJpeg()*, and *Xrt3dSaveImagePng()* are used for this purpose. Please see Appendix C on page 161 for a description of these functions.

Sample Print Pop-up

The source code for a sample print pop-up window which shows one way to use *Xrt3dDrawPS()* and how to pipe the file to *lp* is in *$XRTHOME/src/tools/pdbox_3d.c*. Figure 20 shows the windows created by the sample print pop-up code.

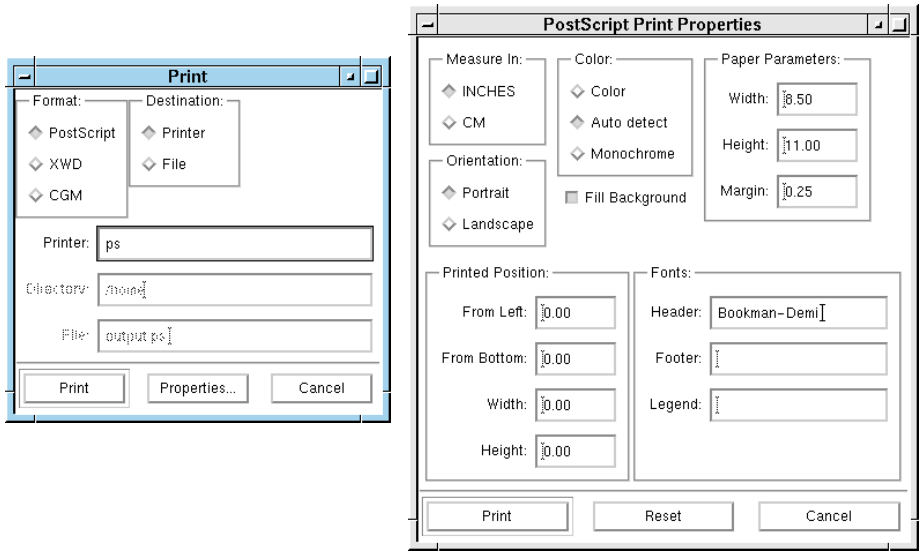


Figure 20 Print Pop-ups created by the sample Print Pop-up Code

4

Programming XRT/3d: Surfaces and Bars

15 Basic Types of Surfaces and Bars ■ *Bar Charts and Histograms*
Contours and Zone Display ■ *Mesh Controls*
Surface Colors ■ *Solid Surface*

4.1 15 Basic Types of Surfaces and Bars

Any grid data (i.e. data of type `XRT_DATA_GRID` or `XRT_DATA_IRGRID`) that has been attached to the graph widget using the `XtNxrt3dSurfaceData` resource will be displayed in a surface, bar or scatter plot representation. The type of representation depends on the value of the `XtNxrt3dType` resource, which can be `XRT3D_TYPE_SURFACE`, `XRT3D_TYPE_BAR` or `XRT3D_TYPE_SCATTER`. For information on scatter plots, see "Programming XRT/3d: Scatter Plots" on page 57.

Note that using a data set of type `XRT3D_DATA_POINT` with a type of `XRT3D_TYPE_SURFACE` or `XRT3D_TYPE_BAR` will produce no results. `XRT3D_DATA_POINT` is only usable with scatter plots.

When grid data is used and the type is either `XRT3D_TYPE_SURFACE` or `XRT3D_TYPE_BAR`, XRT/3d's four basic display boolean resources–DrawMesh, DrawShaded, DrawContours and DrawZones–combine to create 15 different basic surface and bar displays.[1]

DrawMesh

Surfaces: When DrawMesh is `True`, XRT/3d displays the X-Y grid projected onto the 3D surface in a 3D view with a Z-axis. The graph honors rotation and perspective control. The `XtNxrt3dDrawHiddenLines`, `XtNxrt3d[XY]MeshShow` and `XtNxrt3d[XY]MeshFilter` resources are used to fine-tune the mesh display, and

1. No graph is displayed when all four booleans are False. Note that none of the four draw methods affect scatter plots.

XtNxrt3dMeshBottomColor and XtNxrt3dMeshTopColor resources control the mesh colors.

Bars: When DrawMesh is True, XRT/3d will draw the outline of all the bars. All bars with value greater than or equal to XtNxrt3dZOrigin will be outlined using XtNxrt3dMeshTopColor, and all bars with value less than XtNxrt3dZOrigin will be outlined using XtNxrt3dMeshBottomColor. When XtNxrt3dDrawHiddenLines is True, all the lines of every bar are visible.

DrawShaded

Surfaces: When DrawShaded is True, XRT/3d displays the data as a flat shaded surface in a 3D view with a Z-axis. The graph honors rotation and perspective control. The surface color is controlled with XtNxrt3dSurfaceBottomColor and XtNxrt3dSurfaceTopColor.

Bars: When DrawShaded is True, XRT/3d draws each bar as a solid bar. All bars with value greater than or equal to XtNxrt3dZOrigin will be drawn using XtNxrt3dSurfaceTopColor, and all bars with value less than XtNxrt3dZOrigin will be drawn using XtNxrt3dSurfaceBottomColor.

DrawContours

Surfaces: When DrawContours is True, XRT/3d examines the distribution of the data (using the XtNxrt3dDistnMethod and possibly XtNxrt3dDistnTable), and draws contour lines demarcating each of the distribution levels. The contour line style, thickness and color are controlled with the XtNxrt3dContourStyles resource.

Bars: When DrawContours is True, XRT/3d examines the distribution of the data (using XtNxrt3dDistnMethod and possibly XtNxrt3dDistnTable), and draws contour lines around the bars, demarcating each of the distribution levels. The contour line style, thickness and color are controlled with the XtNxrt3dContourStyles resource.

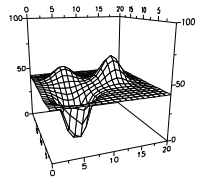
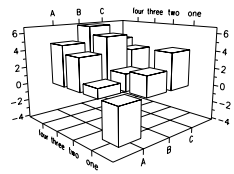
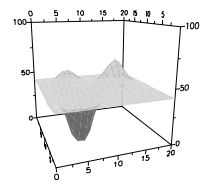
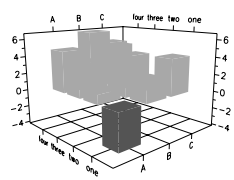
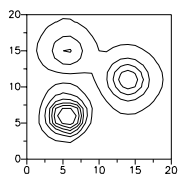
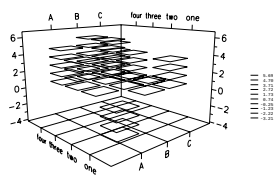
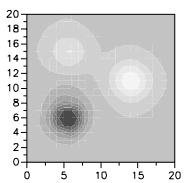
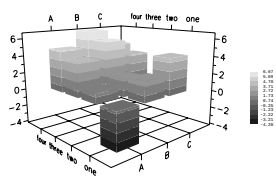
DrawZones

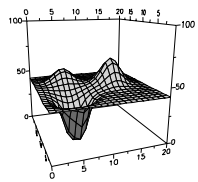
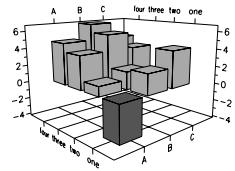
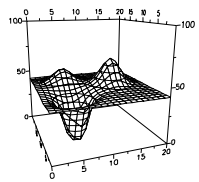
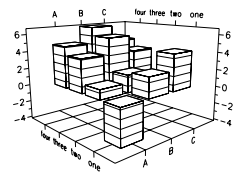
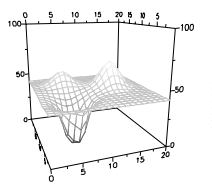
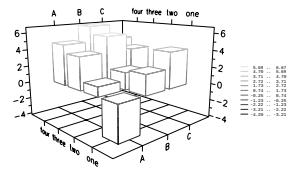
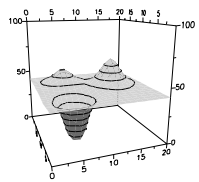
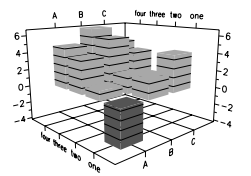
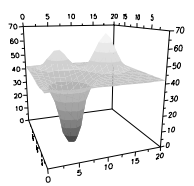
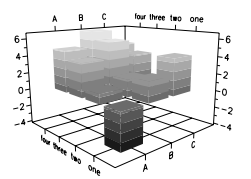
Surfaces: When DrawZones is True, XRT/3d examines the distribution of the data (using the XtNxrt3dDistnMethod and possibly XtNxrt3dDistnTable), and fills each level with a solid color[1]. The color for each level is specified with the XtNxrt3dContourStyles resource.

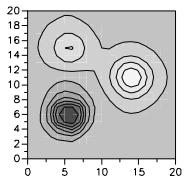
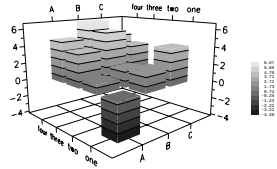
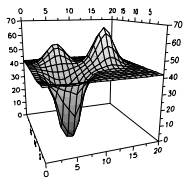
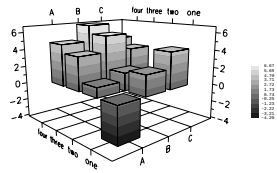
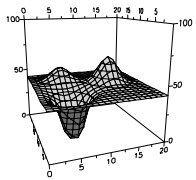
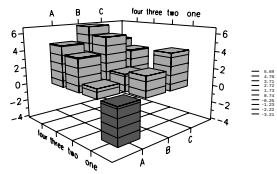
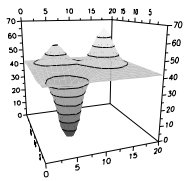
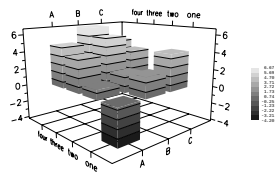
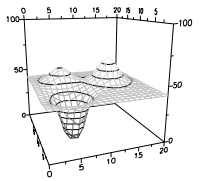
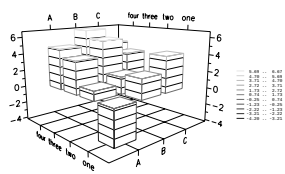
Bars: When DrawZones is True, XRT/3d examines the distribution of the data (using XtNxrt3dDistnMethod and possibly XtNxrt3dDistnTable), and fills each level within each bar with a solid color.[1] The color for each level is specified with the XtNxrt3dContourStyles resource. If XtNxrt3dZoneData is supplied, each bar is filled with a solid color. Otherwise, the bar is segmented by height according to the distribution table.

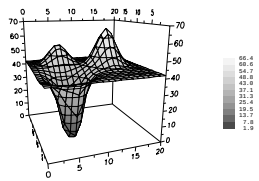
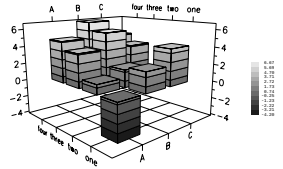
1. Unless DrawMesh is True and DrawShaded is False, in which case the fill color is used to draw each level's mesh lines.

The following table shows the 15 basic graph types:

Graph Type	DrawMesh	DrawShaded	DrawContours	DrawZones	Surface Example	Bar Example	Comments
1	T	F	F	F			**Mesh.** Displays surface as a mesh and bars in outline.
2	F	T	F	F			**Shaded.** Displays surface and bars in a flat shade. Top and bottom colors may be set.
3	F	F	T	F			**Contours.** Contour lines are automatically drawn between distribution levels in the data.
4	F	F	F	T			**Zones.**[a] Similar to #3, except that each distribution level is displayed in a solid color.

Graph Type	DrawMesh	DrawShaded	DrawContours	DrawZones	Surface Example	Bar Example	Comments
5	T	T	F	F			**Mesh, Shaded.** Draws surface as a mesh and bars in outline. Surface and bars are flat shaded.
6	T	F	T	F			**Mesh, Contours.** Displays surface as a mesh and bars in outline. Also draws contour lines along borders between distribution levels in the data.
7	T	F	F	T			**Mesh, Zones.** Displays surface as a mesh and bars in outline. Uses zoning colors for mesh and bar outlines.
8	F	T	T	F			**Shaded, Contours.** Displays a flat-shaded surface or bars with contour lines superimposed.
9	F	T	F	T			**Shaded, Zones.** Zone colors are used to flat shade the surface or bars.

Graph Type	DrawMesh	DrawShaded	DrawContours	DrawZones	Surface Example	Bar Example	Comments
10	F	F	T	T			**Contours, Zones.**[a] Displays contour lines and flat shaded zone colors to demarcate levels in the data.
11	T	T	F	T			**Mesh, Shaded, Zones.** Like #9, but with a mesh or bar outlines superimposed.
12	T	T	T	F			**Mesh, Shaded, Contours.** Like #5, but contour lines are superimposed.
13	F	T	T	T			**Shaded, Contours, Zones.** Like #9, but contour lines are superimposed.
14	T	F	T	T			**Mesh, Contours, Zones.** Like #7, but with contours superimposed.

Graph Type	DrawMesh	DrawShaded	DrawContours	DrawZones	Surface Example	Bar Example	Comments
15	T	T	T	T			**Mesh, Shaded, Contours, Zones.** The sum of all basic options.

a. In this release, DrawZones is the same as DrawZones and DrawShaded for bar charts. Application developers are urged to use the DrawZones and DrawShaded combination for this view, since the interpretation of the DrawZones combination may change in a future release.

4.2 Bar Charts and Histograms

When `XtNxrt3dType` is `XRT3D_TYPE_BAR`, the data pointed to by the `XtNxrt3dSurfaceData` resource will be displayed as a bar chart (if grid data is supplied). Each data point will be represented by a single bar. Data for bar charts may be supplied in either a regular or irregular grid. In either case, the spacing between adjacent elements in the grid is honored.

Bar Z Origin

Bars start from the Z origin, which can be controlled through the `XtNxrt3dZOrigin` resource. The default value of the origin is 0.0. When DrawShaded is `True`, bars that have values greater than the origin are rendered in the `XtNxrt3dSurfaceTopColor`. Negative bars (that is, bars with values less than the origin) are rendered in the `XtNxrt3dSurfaceBottomColor`.

Bar Spacing

The amount of space occupied by a bar, as a percentage of the maximum amount possible, is controlled through the `XtNxrt3d[XY]BarSpacing` resources. The default is 80%. Setting it smaller results in thinner bars. Setting bar spacing to 100% (the maximum) results in bars that abut one another.

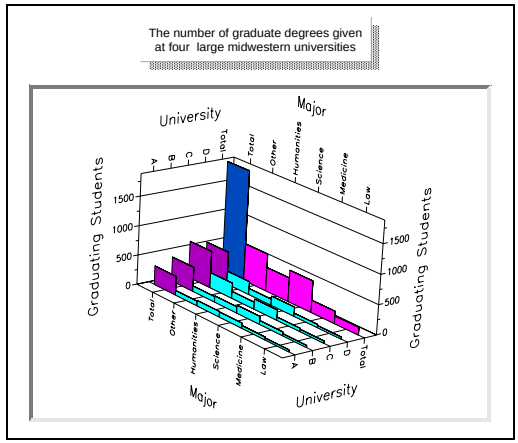


Figure 21 Fixed Bar Chart with X Spacing set to 1% and Y Spacing set to 100%

Histograms

To display a histogram, set `XtNxrt3d[XY]BarFormat` to `XRT3D_BAR_HISTOGRAM`. (The default is `XRT3D_BAR_FIXED`.) The X-axis and Y-axis can be independently switched between fixed and histogram formats. If the X-axis is switched to a histogram, each bar's left edge will be drawn aligned with corresponding X values in the data. The width of each bar will be the distance between subsequent X values in the data. Since the width of each bar is derived from the spacing between its neighbors, there is always one fewer bar along a histogram axis than there is if the same data is displayed along a fixed axis.

Histograms usually make use of irregularly-gridded data. This gives control over spacing in the data grid. See section 6.2 on page 66 for more information on irregularly-gridded data.

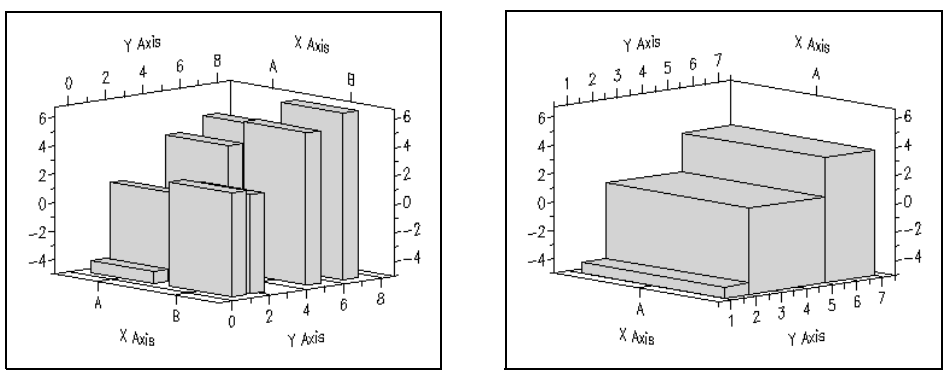


Figure 22 Fixed versus Histogram Display of Identical Data.

Bar Colors

If DrawShaded is `True` and DrawZones is `False`, the colors of the individual bars can be controlled. Normally, all bars with value below the origin are colored with the surface bottom color, and all others are colored with the surface top color. However, in some situations, it is useful to color a row, column or individual bar in the chart with a distinct color.

The resource `XtNxrt3dXYColors` is used to specify the fill color of a line of bars, or of an individual bar. It takes a pointer to a `NULL`-terminated list of pointers to the following structure:

```
typedef struct {
    int             xindex;     /* 0-indexed */
    int             yindex;     /* 0-indexed */
    char *          color;      /* color name */
    unsigned long   res;        /* reserved */
} Xrt3dXYColor;
```

An individual bar can be colored by specifying its indices in the structure. For example, to set the bar in the third X data line, second point to red, specify the values `(2, 1, "red")` in one of the `Xrt3dXYColor` structures. To specify the color of an entire row of bars, set the other index to -1. For instance, the entire fifth X line of bars is set to green with `(4, -1, "green")`. To set the color of all the bars, set both *xindex* and *yindex* to -1.

You can retrieve the color used for a bar by calling the *Xrt3dGetXYColor()* procedure. The *Xrt3dSetXYColor()* procedure makes it easy to change the bar coloring. For example, to change the fourth row of bars (as measured along the Y-axis) to yellow, you could use:

```
Xrt3dSetXYColor(mygraph, -1, 3, "Yellow");
```

XRT/3d will only maintain one entry per (*xindex*, *yindex*) combination. Whenever a second entry for the same indices is supplied, the first entry is removed. Later entries take precedence over earlier entries.

If *color* is set to `NULL` in any `Xrt3dXYColor` structure, it is taken as a request to delete any other entry with the same indices. This is most conveniently used with the procedure *Xrt3dSetXYColors()*, to remove individual color entries.

XYColors are not placed in the legend. This method of coloring bars should only be used if it is not important to label the bars a certain color. If labelling is important, a 4D bar chart should be created. 4D bar charts are discussed in section 8.2 on page 88.

bars.c

Source code for a simple bar chart example program is in *$XRTHOME/src/3d/examples*. It allows the user to change the color of individual bars by clicking on them. When *bars.c* is compiled and run, the window shown in Figure 23 appears.

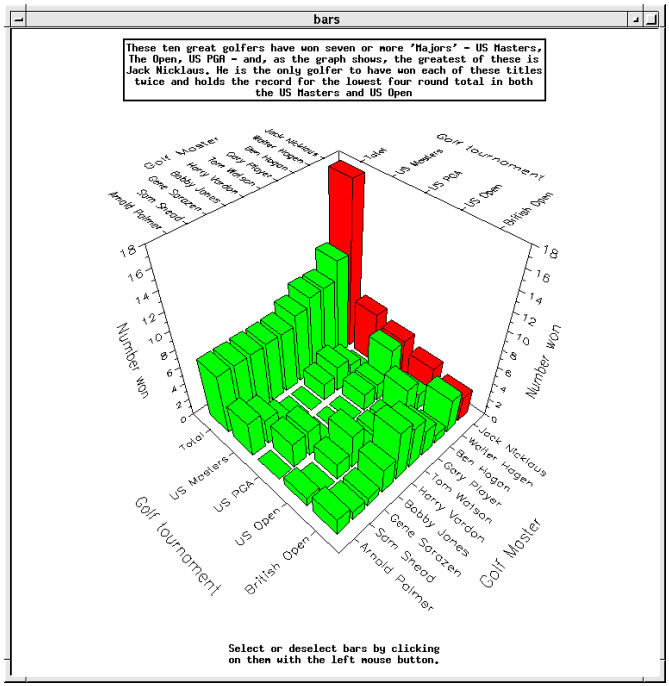


Figure 23 The Window Created by bars.c

4.3 Contours and Zone Display

When DrawContours or DrawZones is `True`, XRT/3d marks each distribution level from an array of 100 built-in contour styles. To present the best-looking display possible, different contour style arrays are used for color and greyscale (includes monochrome) X Servers.

Each contour style contains information about the contour line style, width, pattern, color, and zone color (used to mark each *level*). Contour styles can be customized by an application. For more information, see section 8.6 on page 95.

Contour Styles Used

XRT/3d determines which contour style to use for a particular level automatically, evenly distributing the styles through the number of levels, as shown by Figure 24.

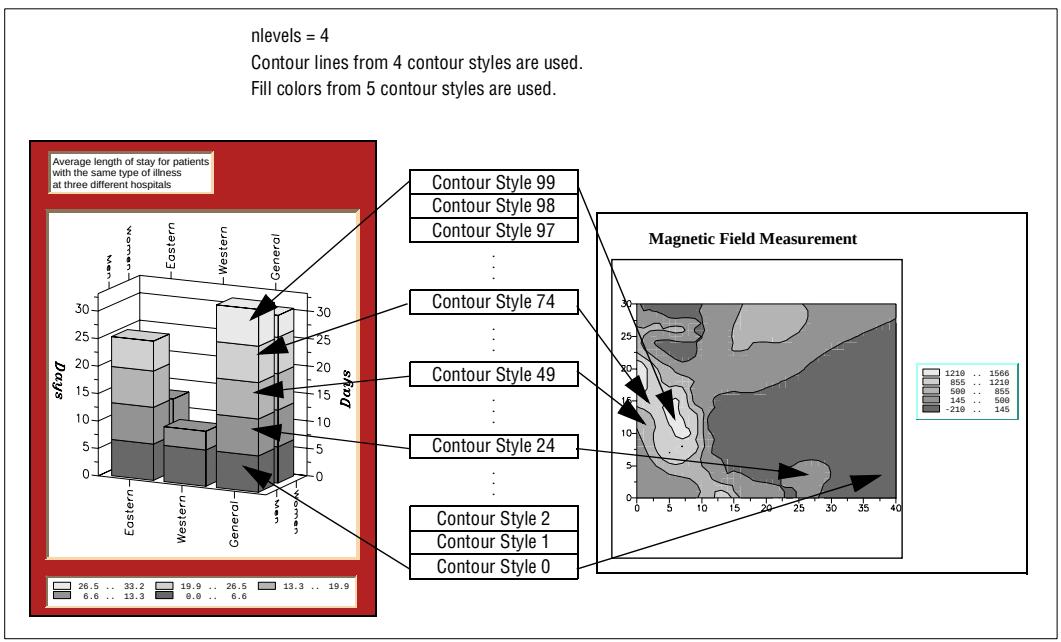


Figure 24 A Sampling of Supplied Contour Styles Used as Needed

Contour/Zone Projection

The contours and zones determined for the graph can be displayed on the top or bottom side of the unit cube using the `XtNxrt3dProjectZMax` and `XtNxrt3dProjectZMin` resources. These resources take integers combined from the following constants:

```
#define XRT3D_PROJECT_CONTOURS  0x1
#define XRT3D_PROJECT_ZONES     0x2
```

For instance, to project contours and zones onto the plane $z=zmax$, set `XtNxrt3dProjectZMax` to the value

(XRT3D_PROJECT_CONTOURS|XRT3D_PROJECT_ZONES). To remove all projections from a side, set the resource to 0.

These resources are not dependent on the values of DrawContours or DrawZones. However, any resource that affects contour generation or rendering (such as XtNxrt3dContourStyles, XtNxrt3dNumDistnLevels or XtNxrt3dZoneData) affects projected contours/zones. XtNxrt3dProjectZMax and XtNxrt3dProjectZMin are ignored in 2D graphs, bar charts and scatter plots.

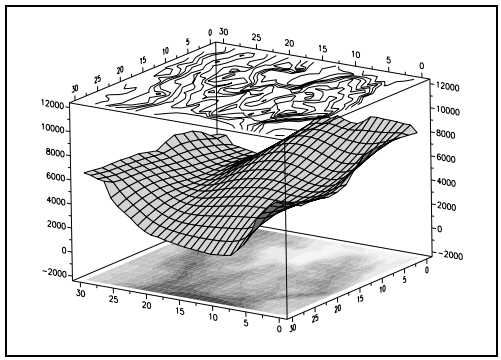


Figure 25 Projecting Contours and Zones

Zoning Method

An application can control the method used to fill each zone region with the XtNxrt3dZoneMethod resource. By default (XRT3D_ZONE_CONTOURS), XRT/3d fills between each contour interval. When set to XRT3D_ZONE_CELLS, XRT/3d fills entire cells in the grid based on the average of the four corners of the cell. Figure 26 illustrates the visual difference.

Cell zoning produces a coarser-looking surface, but offers significant performance advantages over contour zoning. However, the visual difference between the two methods diminishes with larger grids.

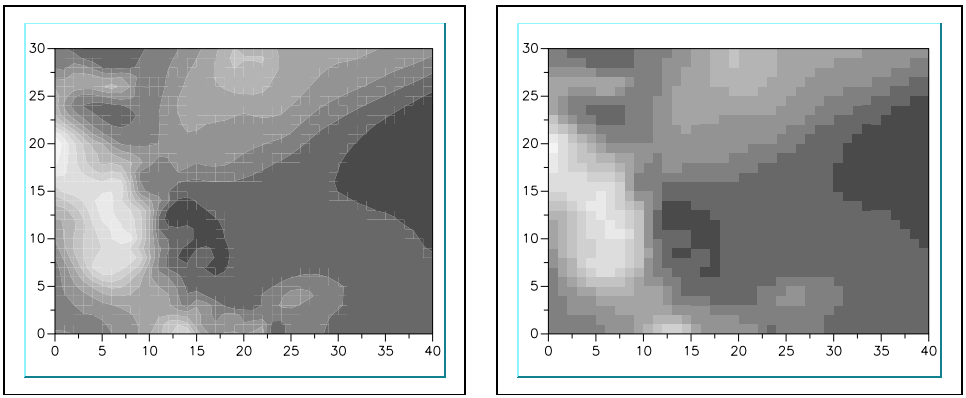


Figure 26 Contour Zoning (left) and Cell Zoning (right)

4.4 Mesh Controls

Mesh Colors

The bottom and top colors of the mesh drawn when DrawMesh is `True` can be set with `XtNxrt3dMeshBottomColor` and `XtNxrt3dMeshTopColor`. They are both "Black" by default. See section 2.7 on page 19 for details on setting colors.

Mesh Filtering

The `XtNxrt3d[XY]MeshFilter` resources specify how the mesh is filtered before being displayed. By default, no filtering is performed. When set to 0, XRT/3d automatically filters the mesh to provide a pleasing display, and changes the filter as the graph is scaled or the data changes.

You can hard-code a mesh filter by setting these resources to any positive integer. For example, a value of 5 filters the mesh so that every 5th line is drawn.

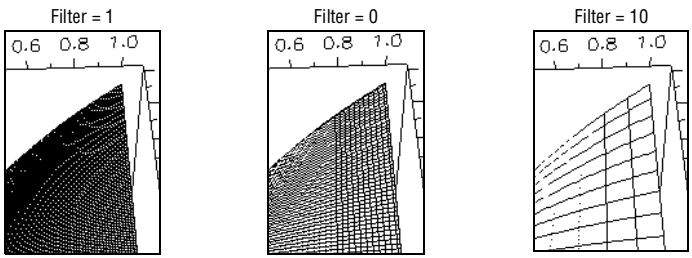


Figure 27 Effect of Mesh Filtering

Hidden Mesh Lines

When DrawMesh is `True` and DrawShaded is `False`, grid and contour lines that are obscured from view by intervening portions of the scene are not displayed by default. To display these lines, set `XtNxrt3dDrawHiddenLines` to `True`.

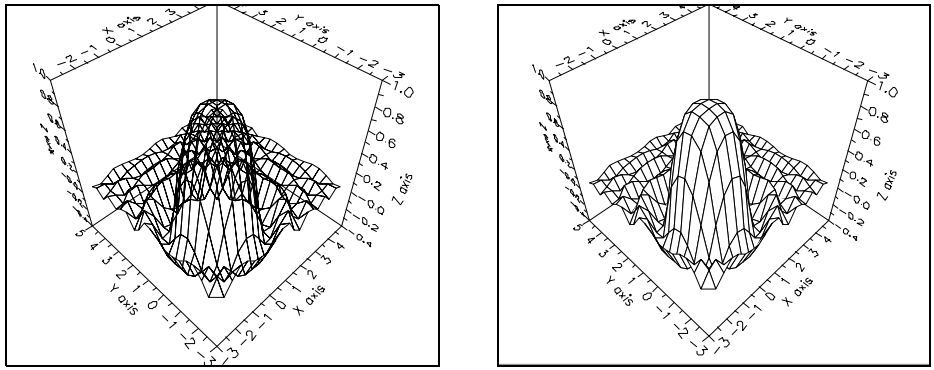


Figure 28 Hidden Line Removal

4.5 Surface Colors

The bottom and top colors of the shaded surface drawn when DrawShaded is `True` can be set with `XtNxrt3dSurfaceBottomColor` and `XtNxrt3dSurfaceTopColor`. By default the bottom color is “dim grey” and the top color is “light grey”. See section 2.7 on page 19 for details on setting colors.

4.6 Solid Surface

Setting `XtNxrt3dSolidSurface` to `True` will cause XRT/3d to draw a skirt around the data, thereby joining the edge of the surface to a plane at the minimum Z value, as shown in Figure 29.

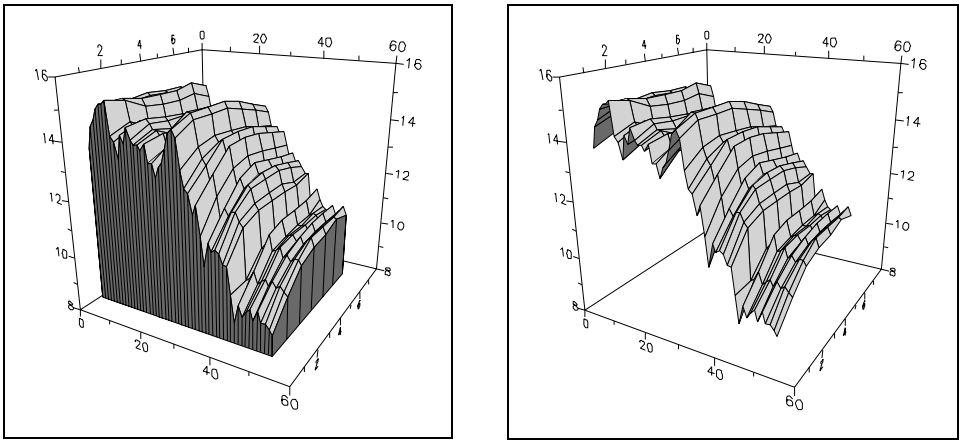


Figure 29 Setting Solid Surface On and Off

5

Programming XRT/3d: Scatter Plots

3 Basic Types of Scatter Plots

Controlling Symbol and Drop Line Style ■ Data Styles

5.1 3 Basic Types of Scatter Plots

Any data that has been attached to the graph widget using the `XtNxrt3dSurfaceData` resource, where the value of the `XtNxrt3dType` resource is `XRT3D_TYPE_SCATTER` will be displayed as a scatter plot. Scatter plots display one or more series of (x, y, z) points. The usual way of specifying scatter plot data is to use the `XRT3D_DATA_POINT` type, referred to as *point data* in this manual. This type supports an arbitrary number of series', each containing an arbitrary number of points.

Each series can have a symbol that makes it distinct from the other series'. When `XRT3D_TYPE_SCATTER` is used in conjunction with grid data (a data set of type `XRT_3D_DATA_GRID` or `XRT3D_DATA_IRGRID`), each grid point will define a point in the scatter plot. Only one data series will be used in this case.

XRT/3d offers three basic types of scatter plots: 3D scatter plots, 3D scatter plots with drop lines, and 2D scatter plots.

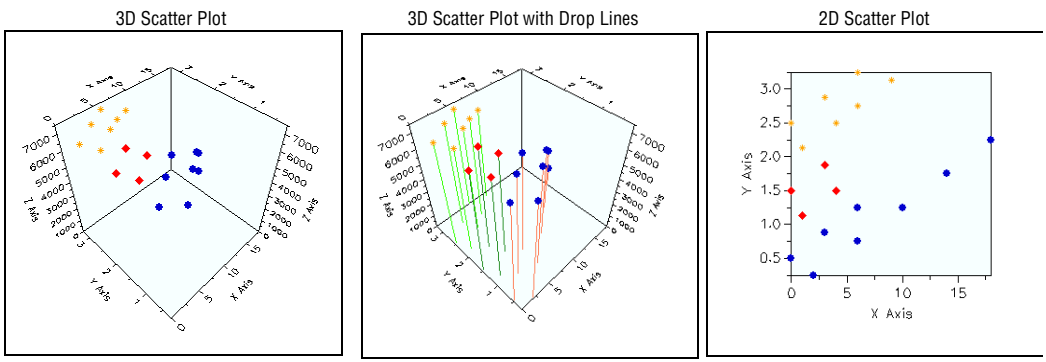


Figure 30 Scatter Plot examples

Drop lines are lines drawn from each point on the 3D scatter plot down to the matching (x,y) position on the z-axis origin of the plot cube. The drop line joins the point (x, y, z) to the matching point (x, y, origin). Drop lines are not used for 2D scatter plots. To draw drop lines on 3D scatter plots, set `XtNxrt3dDrawDropLines` to `True`.

2D scatter plots use only the x and y values, and ignore the z components. They are "flat" charts. These plots are created by setting both `XtNxrt3dDrawMesh` and `XtNxrt3dDrawShaded` to `False` (the same method that is used for surface and bar charts).

5.2 Controlling Symbol and Drop Line Style

The style of symbols and drop lines used in scatter plots is determined by the data type `Xrt3dDataStyle`, which specifies the line and symbol style information for a series. One data style must be defined for each series, since each series matches up with its corresponding data style. For example, the first series uses the first data style (with index 0), the second series uses the second data style, and so on. If fewer data styles are defined than are needed for the number of series', then the default data styles will be used for the remaining series'.

Line pattern, color and width are controlled by `Xrt3dLineStyle` and symbol color, size and pattern by `Xrt3dSymbolStyle`.

5.3 Data Styles

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

Default Data Styles

XRT/3d provides default data styles for monochrome and color Servers. If `XtNxrt3dDataStylesUseDefault` 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 `XtNxrt3dDataStylesUseDefault` to `True`.

The Xrt3dDataStyle structure is declared in *<Xm/xrt3dcom.h>*:

```
typedef struct {
    Xrt3dLinePattern    pattern;/* line pattern */
    XrtColor            color;  /* color */
    int                 width;  /* line width */
} Xrt3dLineStyle;

typedef struct {
    Xrt3dSymbolPattern  pattern;/* symbol pattern */
    XrtColor            color;  /* color */
    int                 size;   /* symbol size */
} Xrt3dSymbolStyle;

typedef struct {
    Xrt3dLineStyle      line_style; /* line style */
    Xrt3dSymbolStyle    symbol_style;/* symbol style */
} Xrt3dDataStyle;
```

The Xrt3dDataStyle data structure contains all the information about how a set of data will be represented graphically. The fields are broken down as follows:

pattern	The line pattern used for drop lines or symbol pattern used for scatter plot points as defined in *<Xm/xrt3dcom.h>*. For line patterns, must be one of the XRT3D_LPAT constants listed in Figure 31. For scattter symbol patterns, must be one of the XRT3D_SYMBOL constants listed in Figure 32.
color	The color used when drawing drop lines or scatter symbols. 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 2.7 on page 19 for details of color specification.
width	The line width used for drop lines. Must be greater than or equal to 1.
size	The size is the diameter in pixels of the scatter symbol. It must be greater than or equal to 1.

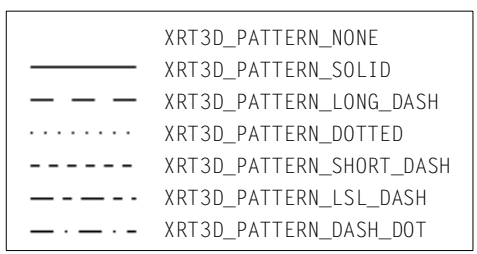


Figure 31 Line Patterns

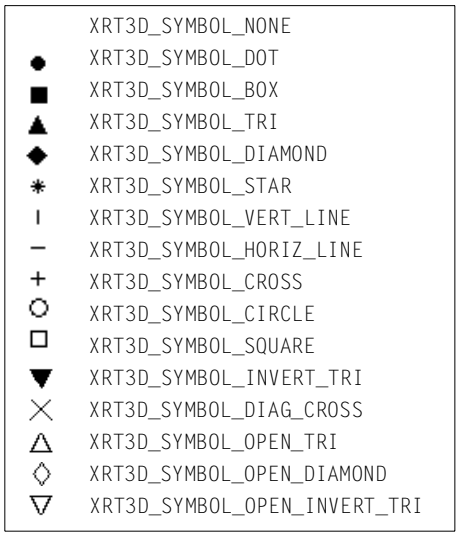


Figure 32 Symbol Styles

Explicitly Setting Data Styles

It is usually satisfactory to use XRT/3d'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 `XtNxrt3dDataStylesUseDefault` resource will be set to `False`. XRT/3d 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 33.

```
static Xrt3dDataStyle scatter_styles[] = {
/* 0 */ {{ XRT3D_LPAT_SOLID,        "coral",          1 },
         { XRT3D_SYMBOL_DOT,        "medium blue",    7 }},
/* 1 */ {{ XRT3D_LPAT_SOLID,        "forest green",   1 },
         { XRT3D_SYMBOL_DIAMOND,    "red",            7 }},
/* 2 */ {{ XRT3D_LPAT_SOLID,        "lime green",     1 },
         { XRT3D_SYMBOL_STAR,       "orange",         7 }},
/* 3 */ {{ XRT3D_LPAT_SOLID,        "royal blue",     1 },
         { XRT3D_SYMBOL_TRI,        "magenta",        7 }}
};

static Xrt3dDataStyle *my_dstyles_ptr[] = {
  &scatter_styles[0],
  &scatter_styles[1],
  &scatter_styles[2],
  &scatter_styles[3]
};

....

XtVaSetValues(graph, XtNxrt3dDataStyles, my_dstyles_ptr, NULL);
```

Figure 33 Setting Data Styles

Accessing Data Styles

Data styles are accessed through the `XtNxrt3dDataStyles` resource and/or through the *Xrt3dSetNthDataStyle()* and *Xrt3dGetNthDataStyle()* methods.

`XtNxrt3dDataStyles` 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 *Xrt3dDupDataStyles()* convenience routine may be used to duplicate the returned data styles array pointer. For example, to set all drop line widths to 5-pixels thick, the following code could be used:

```
void SetThickLines(Widget graph)
{
   int i;
    Xrt3dDataStyle **ds, **myds;
    XtVaGetValues (graph, XtNxrt3dDataStyles, &ds, NULL);
    myds = Xrt3dDupDataStyles(ds);
    for(i=0; myds[i]; i++) {
        myds[i]->line_style.width = 5;
    }
    XtVaSetValues(graph, XtNxrt3dDataStyles, myds, NULL);
    /* Release this copy once styles are set */
    Xrt3dFreeDataStyles(myds);
}
```

A program can get a pointer to a particular `Xrt3dDataStyle` structure using *Xrt3dGetNthDataStyle()*. For example, if `graph` is a plot of scatter data, the following code will print out the symbol size being used for the series of data, and double it:

```
void DoubleSymbolSize(Widget graph, int series)
{
   Xrt3dDataStyle *dstyle, my_dstyle;
   dstyle = Xrt3dGetNthDataStyle(graph, series);
   printf("line width for series %d of points: %d\n", series,
       dstyle->symbol_size.size);
/* Treat dstyle as a "read-only" ptr. Modify a local copy */
   my_dstyle = *dstyle;
   my_dstyle.symbol_style.size *=2;
   Xrt3dSetNthDataStyle(graph, series, &my_dstyle);
}
```

6

XRT/3d Data

Data Overview
The Xrt3dData Structure

6.1 Data Overview

Data to be graphed can originate from diverse sources: Unix files, databases, real time data feeds, or even unrelated processes running on other machines. The data can represent the results of an experiment or the solution to an analytical problem. XRT/3d displays data graphically. It does not do any analysis of the data, except for possibly displaying its zone and contour distributions.

This chapter discusses the `Xrt3dData` structure in detail and offers examples of allocating and loading the `Xrt3dData` structure.

Rules and Guidelines

XRT/3d graphs surfaces that are increasing in X and Y. It does not graph surfaces that fold back in X or Y (such as a sphere).

For surfaces and bars, XRT/3d expects its data to come from a regularly or irregularly gridded surface. Think of X and Y gridlines forming a mesh. When gridlines meet, they define a point which has a Z value. If a point doesn't have a Z value, it is called a *hole*.

For scatter plots, XRT/3d will also accept data provided through one or more series' of (x, y, z) points.

XRT/3d is optimized for small dimension meshes. Very large dimension data will take longer to graph, and the results may not be satisfactory. If you need to graph large dimension data, thin it out before using it with XRT/3d. For example, you might decide to load every tenth grid point into the `Xrt3dData` structure or use the *Xrt3dDataWindow()* procedure.

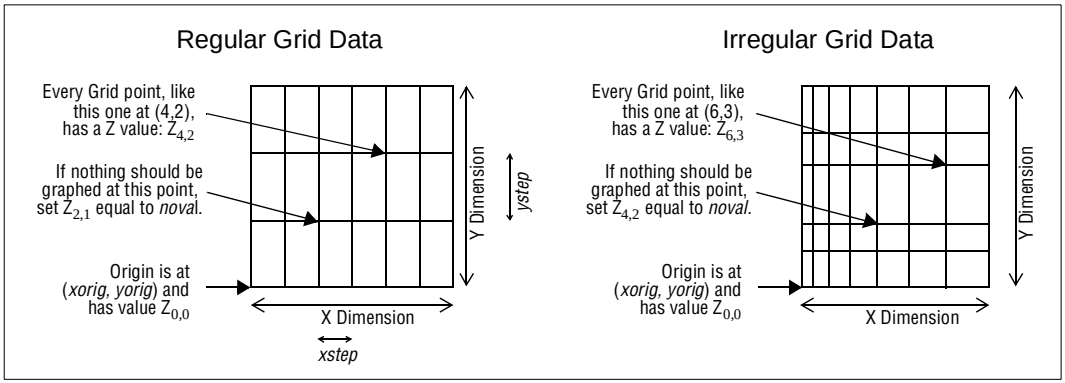


Figure 34 7 x 4 Regular grid and 8 x 6 Irregular grid

Holes in the Data

The `Xrt3dData` structure contains an element called *noval*. Whenever XRT/3d sees values in the grid data that are the same as *noval*, it will treat that grid intersection as a hole. You should set *noval* to be a value that is well outside of the range of your data. For example, you may want to set it to `XRT3D_HUGE_VAL`.

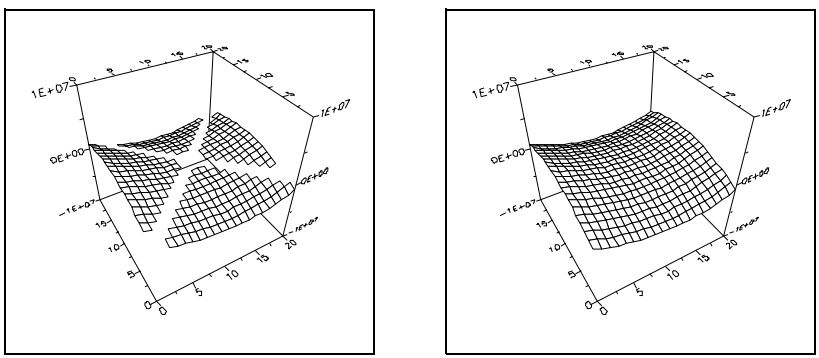


Figure 35 Example of a Hole. (Graph on left has noval set to 0.)

Data from a File

If the data exists in a 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 *Xrt3dMakeDataFromFile()*. This procedure will allocate the `Xrt3dData` structure and load it with data from a named file. *Xrt3dMakeDataFromFile()* is fully described in Appendix C. There is also a corresponding *Xrt3dSaveDataToFile()* procedure.

Another approach is to allocate an `Xrt3dData` structure using *Xrt3dMakeGridData()*, *Xrt3dMakeIrGridData()* or *Xrt3dMakePointData()* and populate it with data by reading the data file (perhaps by using *fgets(3)* or *fscanf(3)*).

Changing Data

It is easy to change data while leaving other widget resources unchanged. Programs can create any number of `Xrt3dData` structures and then switch among them by setting the `XtNxrt3dSurfaceData` resource to point to the current data.

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 call-back routines to deal with real time events.

In particular, *XtAppAddInput()* can be used to register an application call-back 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 *XtAppAddTimeOut()* to define an interval timer. Whenever the specified time interval expires, the applications call-back 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.

Real Time Performance

For the best possible real time performance, try to keep the grid dimensions small. For surfaces and bars, DrawShaded and DrawZones are the slowest types of displays and DrawMesh is the fastest type of display. If `XtNxrt3dDrawHiddenLines` is `True`, XRT/3d uses an optimized algorithm to speed display even further. Keeping the window size small and turning off double buffering will also improve performance. Use of `XtNxrt3dRepaint` can also improve performance dramatically.

Responsibility for Data

It is the application's responsibility to allocate and destroy all required `Xrt3dData` structures. XRT/3d does not make a copy of the data; instead, it references data in the application's memory pointed to by `XtNxrt3dSurfaceData` and by `XtNxrt3dZoneData`.

If XRT/3d has to repaint the widget window, it will use the data pointed to by `XtNxrt3dSurfaceData`. An application may adjust the values inside an `Xrt3dData` structure while that structure is in use, if it can ensure that the structure will be correct when next required by XRT/3d. Remember to reset the `XtNxrt3dSurfaceData` resource after changing any data values so that XRT/3d refreshes the display.

6.2 The Xrt3dData Structure

Whenever XRT/3d displays a graph image, it uses the data values located in the `Xrt3dData` structure pointed to by the `XtNxrt3dSurfaceData` resource.

The `Xrt3dData` structure is defined in *<Xm/xrt3d_com.h>*. (It is also defined in Appendix E.) The `Xrt3dData` structure is defined as a union of three data structures:

```
typedef union{
    Xrt3dGridData   g; /* regular grid */
    Xrt3dIrGridData ig; /* irregular grid */
    Xrt3dPointData  p; /* scatter plot*/
} Xrt3dData;
```

6.2.1 Regular Grid Data

The `Xrt3dGridData` structure is defined as follows:

```
typedef struct {
    Xrt3dDataType   type; /*XRT3D_DATA_GRID*/
    int             numx, numy;
    double          noval;
    double          xstep, ystep;
    double          xorig, yorig;
    double          **values;
}Xrt3dGridData;
```

The `Xrt3dGridData` structure contains a `type` field which is always initialized to `XRT3D_DATA_GRID` by *Xrt3dMakePointData()* and *Xrt3dMakeDataFromFile()*.

numx and *numy* are integers defining the number of grid lines (i.e. elements) in the X and Y dimensions. For surface and histogram graphs, *numx* and *numy* must be ≥ 2. For bar graphs, *numx* and *numy* must be ≥ 1.

The special value which XRT/3d treats as "no value" is specified by *noval*. The distance between grid lines in the X and Y direction is specified by *xstep* and *ystep*, and should always be > 0. The origin of the entire grid is specified with *xorig* and *yorig*.

values is a two-dimensional array of the Z values for each grid point in X major order, as Figure 36 illustrates.

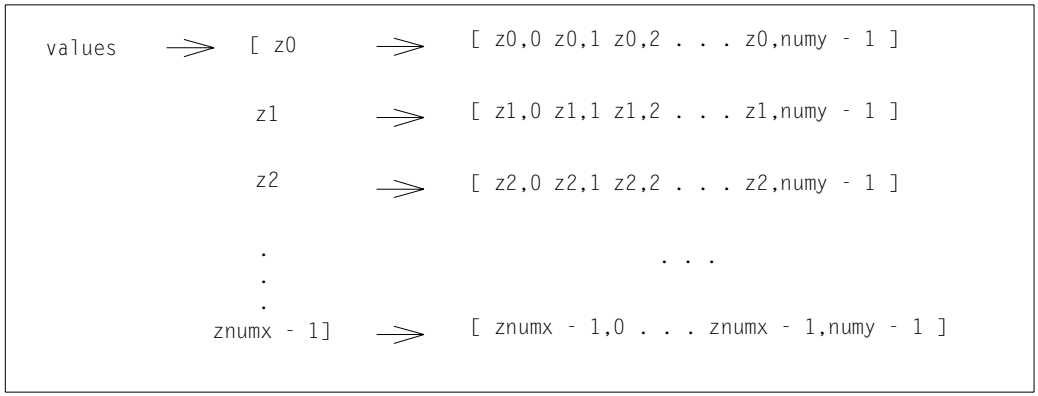


Figure 36 The Two Dimensional Array of Z Values

The easiest way to allocate the `Xrt3dData` structure is by using *Xrt3dMakeGridData()*. This procedure is described in Appendix C.

6.2.2 Irregular Grid Data

The `Xrt3dIrGridData` structure is defined as follows:

```
typedef struct {
    Xrt3dDataType   type; /*XRT3D_DATA_IRGRID*/
    int             numx, numy;
    double          noval;
    double          *xgrid, *ygrid;
    double          **values;
}Xrt3dIrGridData;
```

The `Xrt3dGridData` structure contains a `type` field which is always initialized to `XRT3D_DATA_GRID` by *Xrt3dMakeIrGridData()* and *Xrt3dMakeDataFromFile()*.

numx and *numy* are integers defining the number of grid lines (i.e. elements) in the X and Y dimensions. For surface and histogram graphs, *numx* and *numy* must be ≥ 2. For bar graphs, *numx* and *numy* must be ≥ 1.

The special value which XRT/3d treats as "no value" is specified by *noval*. *xgrid* and *ygrid* are arrays of doubles of length *numx* and *numy* which contain the values of the X and Y grid lines.

values is a two-dimensional array of the Z values for each grid point in X major order.

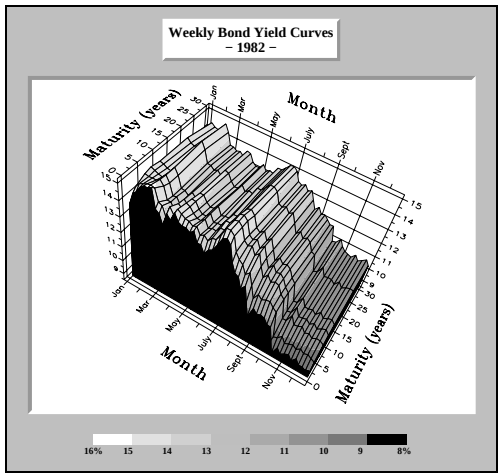


Figure 37 An Irregular Surface Plot

Example of Setting Values (Grid Data)

An application can easily access the `Xrt3dData` structure to read or change values. The following example initializes a regular grid surface to 42.0:

```
int         i, j;
Xrt3dData   *data;

data = Xrt3dMakeGridData(...);

for (i = 0; i < data->g.numx; i++)  {
    for (j = 0; j < data->g.numy; j++) {
        data->g.values[i][j] = 42.0;
    }
}
```

6.2.3 Scatter Plot Data

The `Xrt3dPointData` structure is defined as follows:

```
typedef struct {
    Xrt3dDataType       type; /* XRT3D_DATA_POINT */
    int                 nseries;
    double              noval;
    Xrt3dPointSeries    *series;
} Xrt3dPointData;
```

The `Xrt3dPointData` structure contains a *`type`* field which is always initialized to `XRT3D_DATA_POINT` by *Xrt3dMakePointData()* and *Xrt3dMakeDataFromFile()*.

nseries specifies the number of series' in the point data structure and should always be ≥ 1. The special value which XRT/3d treats as "no value" (i.e. a hole value) is

specified by *noval. series* is an array of nseries' `Xrt3dPointSeries` structures, one for each series. The `Xrt3dPointSeries` structure is defined as follows:

```
typedef struct {
    int             npoints;
    Xrt3dPoint3D    *points;
} Xrt3dPointSeries;
```

npoints is the number of points in the series. *points* is an array of npoints' `Xrt3dPoint3D` structures, one for each point. The `Xrt3dPoint3D` structure is defined as follows:

```
typedef struct {
    double          x, y, z;
} Xrt3dPoint3D;
```

Here is some example code that allocates a `Xrt3dPointData` structure. This example uses the convenience functions *Xrt3dMakePointData()* and *Xrt3dMakePointDataSeries()* to do the allocation:

```
/* Create all of the series */
Xrt3dPointData* pPointData = (Xrt3dPointData *)
Xrt3dMakePointData(nNumSeries, 1, XRT3D_HUGE_VAL, FALSE);

/* Allocate the points in a specific series */
Xrt3dMakePointDataSeries(&(pPointData->series[nSeries])
                         , nNumPoints);
```

6.2.4 Convenience Procedures

There are a number of `Xrt3dData` structure manipulation convenience procedures included with XRT/3d. These procedures are fully described in Appendix C on page 161. The source code for all of the convenience procedures is located in *$XRTHOME/src/3d/util*.

Xrt3dDataCopy()	Creates a copy of an `Xrt3dData` structure.
Xrt3dDataShaded()	Creates a new `Xrt3dData` structure from an existing structure and fills it with a shaded relief map of the original data. An example of this routine can be found in the *shader* demo. Does not apply to point data.
Xrt3dDataSmooth()	Smooths the data in an `Xrt3dData` structure using center-weighted averaging. Does not apply to point data.
Xrt3dDataUpdateDataValue()	Changes a data value to a new value throughout an `Xrt3dData` structure.
Xrt3dDataWindow()	Creates a new `Xrt3dData` structure from an existing structure, extracting a subset of the original data and resampling it using either cubic or linear splines. Does not apply to point data. An example of this routine can be found in the *spline* demo.

7

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

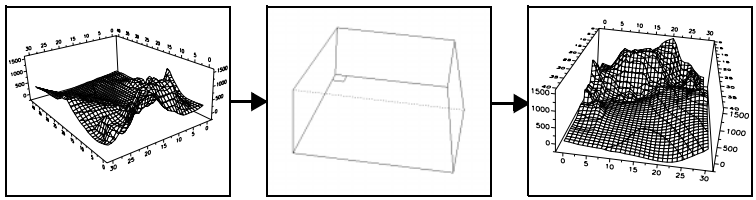
This chapter describes the user-interaction features of XRT/3d–how a user can interact with the graph and how an application can control interaction.

7.1 Default User Interaction

By default, an end-user can rotate, translate, scale, and zoom into the unit cube. An application can also translate a graph programmatically. Figure 38 shows the user interactions enabled by XRT/3d's default translations.

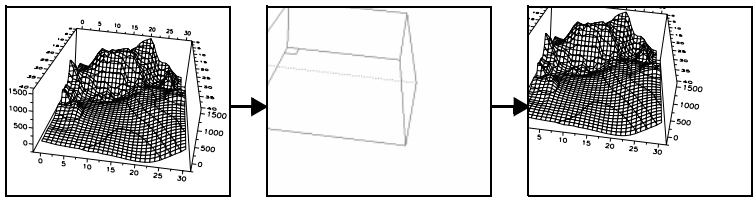
Rotation

- Hold down MB2 and either:
- Move mouse counterclockwise to rotate view clockwise *or*
- Press x, y, z, or e to select an axis, and then move mouse perpendicular to axis



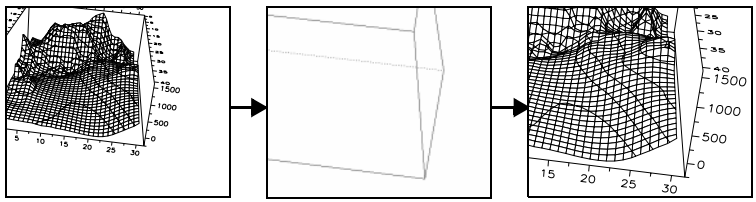
Translation

- Press Shift and hold down MB2
- Move mouse to shift the graph



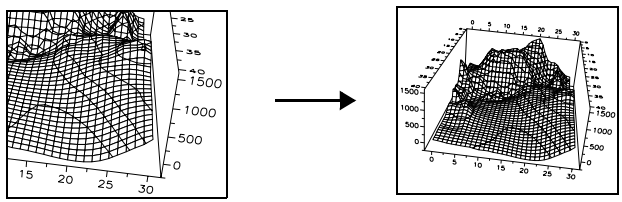
Scaling

- Press Ctrl and hold down MB2
- Move mouse down to zoom in
- Move mouse up to zoom out



Return to Default

- Press “r”
- All scaling, translation, and zooming removed



Zooming

- Press Ctrl and hold down MB1
- Move mouse to select the area to zoom into

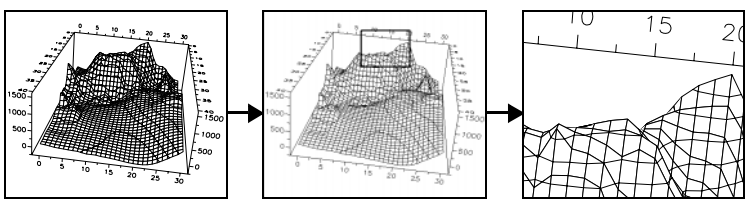


Figure 38 XRT/3d’s Default User Interactions

7.2 Overview of Translations and Actions

XRT/3d's default translation table defines user events that cause some interactive action within the widget. You can customize user interaction through the following mechanisms:

- Translation tables–An application can change the default translation events or actions, or add translations for new actions to the widget. Appendix D on page 187 lists XRT/3d's default translations and actions.
- Callbacks–An application can be notified as a user interacts with the widget by registering callback procedures that are called before, during, and after user interaction. A callback routine can affect each interaction in such ways as disallowing or constraining it. See the section on each interaction for details on using its callback.

Three Interaction Stages

XRT/3d's default user interaction passes through three stages:

- Starting user interaction
- Updating user interaction, and
- Ending user interaction

An interaction must pass through these stages in sequence, and an application can be notified by callbacks during each stage. Each stage is described in the following sections.

7.3 Starting User Interaction

The *ModifyStart()* action is called to notify the application that a user interaction is about to begin. Figure 39 illustrates this action routine and shows where any Modify callbacks you add are called.

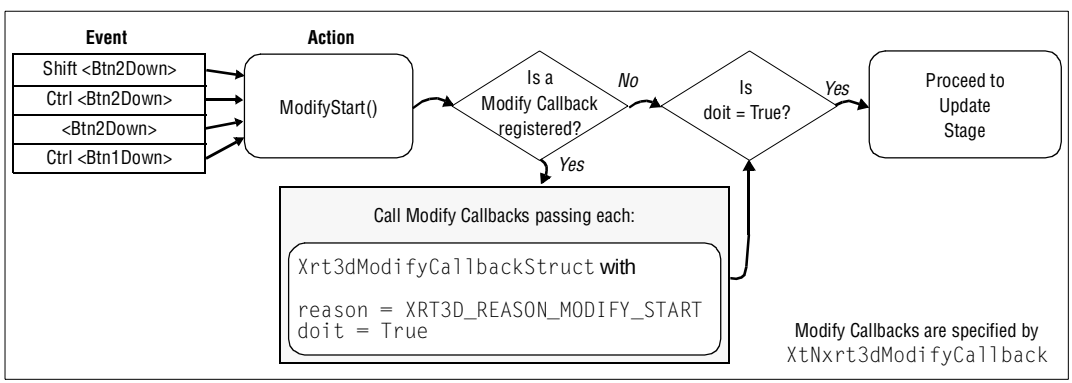


Figure 39 ModifyStart() Action Routine

Disabling All User Interaction

You can use `XtNxrt3dModifyCallback` 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;
} Xrt3dModifyCallbackStruct;
```

Set the *doit* parameter of this structure to `False` (when the *reason* parameter is `XRT3D_REASON_MODIFY_START`)to disable all user interactions. When *doit* is `False`, *ModifyStart()* disables all "update" actions until the next *ModifyStart()* action is called.

7.4 Updating User Interaction

One of several action routines is called to notify the application that a user interaction is about to be updated. No action can update the graph unless the interaction has successfully passed through the *ModifyStart()* action.

DrawPreview() or DrawGraph()

As a user rotates, scales, or translates a graph, XRT/3d can provide visual feedback using either the *DrawPreview()* or *DrawGraph()* action routines. *DrawPreview()* is called by default–it draws a quick outline of the unit cube. An application can redefine a translation to call *DrawGraph()*, which draws the entire unit cube. *DrawGraph()* works best with small datasets.

7.4.1 Scaling, Translation, and 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 alter the `XtNxrt3dViewScale` and `XtNxrt3dView[XY]Translate` resources. Figure 40 illustrates these action routines and shows where any Transform callbacks you add are called.

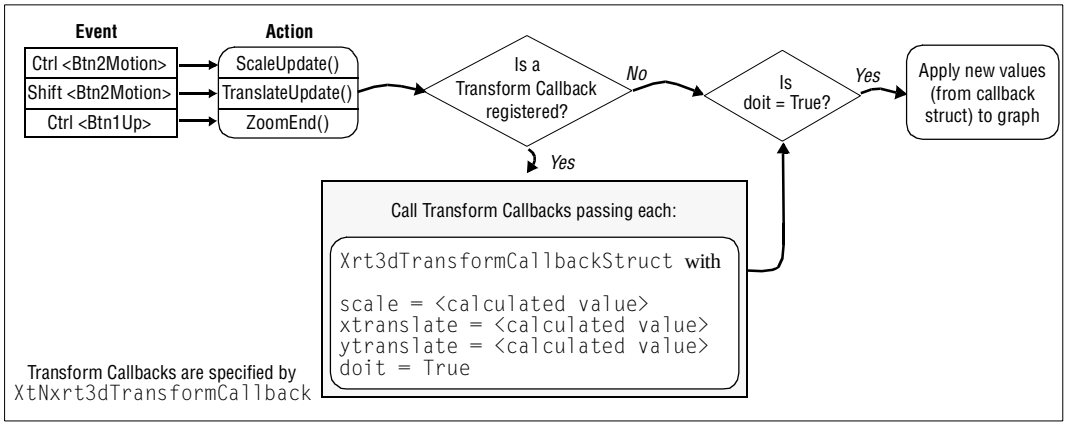


Figure 40 ScaleUpdate(), TranslateUpdate() and ZoomEnd() Action Routines

Controlling Interaction

You can use `XtNxrt3dTransformCallback` 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 */
    double      scale;
    double      xtranslate;
    double      ytranslate;
    Boolean     doit;
} Xrt3dTransformCallbackStruct;
```

A Transform callback can change the *scale*, *xtranslate*, *ytranslate*, and *doit* parameters. *ScaleUpdate()*, *TranslateUpdate()*, and *ZoomEnd()* apply these parameters to the graph. For example, to constrain scaling, examine and change the *scale* parameter in a Transform callback as shown below:

```
void
limit(Widget mygraph, XtPointer cd, XtPointer cbs)
{
    Xrt3dTransformCallbackStruct *tcbs = (Xrt3dTransformCallbackStruc

    if (cbs->scale   < 0.5)cbs->scale   = 0.5;
    if (cbs->scale   > 2.0)cbs->scale   = 2.0;
}
```

Figure 41 Callback procedure that limits scaling between .5 and 2.0

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

```
XtAddCallback(mygraph,
        XtNxrt3dTransformCallback, limit, NULL);
```

Resetting Interactions

The *ViewReset()* action removes all scaling and translation by restoring `XtNxrt3dViewScale` and `XtNxrt3dView[XY]Translate` to their default values.

ViewReset() triggers any registered Transform callbacks, passing each a pointer to an `Xrt3dTransformCallbackStruct` with *doit* set to `True`. An application can deny a reset by setting *doit* to `False` in a callback.

7.4.2 Rotation

The *RotateUpdate()* action updates interactive rotation of the graph. This routine alters the `XtNxrt3d[XYZ]Rotation` resources. Figure 42 illustrates this action routine and shows where any Rotate callbacks you add are called.

XtNxrt3dViewNormalized

By default, XRT/3d resizes the unit cube so that the entire graph is visible (including axis titles and annotation); this setting causes jerky interactive rotation. To allow a user to smoothly rotate a graph, `XtNxrt3dViewNormalized` should be set to `False` before interactive rotation begins.

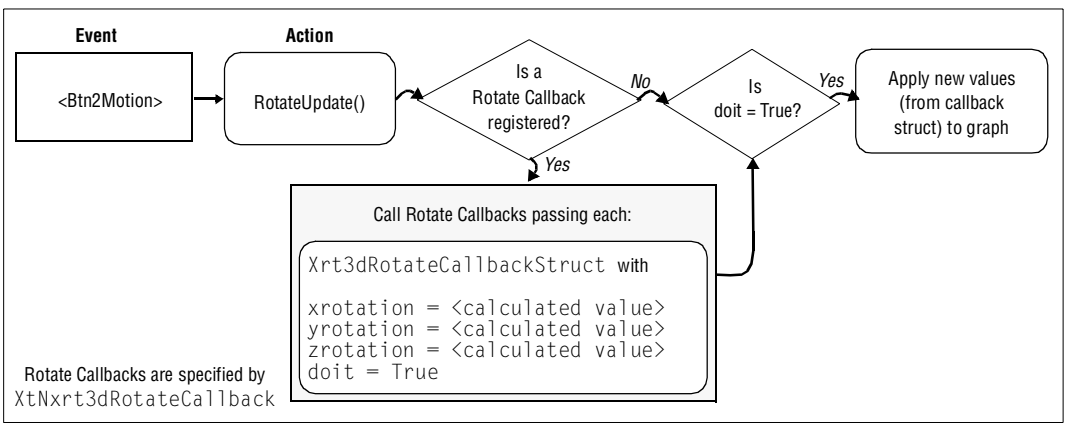


Figure 42 RotateUpdate() Action Routine

Controlling Rotation

You can use `XtNxrt3dRotateCallback` 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 */
    double      xrotation;
    double      yrotation;
    double      zrotation;
    Boolean     doit;
} Xrt3dRotateCallbackStruct;
```

A Rotate callback can change the *xrotation*, *yrotation*, *zrotation*, and *doit* parameters. *RotateUpdate()* applies these parameters to the graph. For example, the following callback disables Y-axis rotation (used in the *view_dir* demo, this makes rotation easier for some end-users to master):

```
void
spin_and_tilt(
    Widget      widget,
    XtPointer   data,
    XtPointer   cbs
{
    Xrt3dRotateCallbackStruct *rotate_cbs;
    double      ay;

    rotate_cbs = (Xrt3dRotateCallbackStruct *) cbs;

    XtVaSetValues(widget,
              XtNxrt3dYRotation, &ay,
              NULL);

    rotate_cbs->yrotation = ay;
}
```

7.4.3 Disabling and Disallowing Interactions

The easiest way to completely *disable* one particular kind of user interaction is to set its translation to *NoAction()* (see section 7.3 for details on disabling all interactions). The following code disables rotation and translation:

```
*.XtXrt3d.translations: #override \n\
    ~Shift ~Ctrl    <Btn2Motion>:      NoAction() \n\
    Shift           <Btn2Motion>:      NoAction()
```

Use a callback procedure to *disallow* an interaction 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 39 and Figure 42). 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 disallows a user's attempt to scale the graph by setting the `Xrt3dTransformCallbackStruct`'s *doit* parameter to `False`:

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

    if (cbs->scale < 0.1){
        cbs->doit = False;
    }
}
```

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

```
XtAddCallback(mygraph,
    XtNxrt3dTransformCallback, deny_small_scale, NULL);
```

7.5 Ending User Interaction

The *ModifyEnd()* action is called to redraw the widget (triggers Expose callbacks) and to notify the application that a user interaction has finished. An application can receive this notification by using `XtNxrt3dModifyCallback`. Each callback procedure is passed a pointer to an `Xrt3dModifyCallbackStruct` structure with the *reason* parameter set to `XRT3D_REASON_MODIFY_END`. Changing the parameters in the structure has no effect.

7.6 Interacting with Graph Data

XRT/3d provides two different ways to handle user interaction with graph data: through action routines/callbacks (section 7.6.1), or directly through event handling procedures (section 7.6.2). Using the action routines is generally easier to program.

7.6.1 Action Routines and Callbacks

You can use a translation table to notify your application when a user attempts to interact with the graph data. The application receives either the graph coordinates clicked or the closest data values to the user event. XRT/3d 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 *Xrt3dPick()* and *Xrt3dMap()* procedures.

Section D.2 on page 188 fully describes XRT/3d'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 *Xrt3dPick()* procedure. You can use the *Pick()* action to find out which set and point a user has clicked nearest. For more information on *Xrt3dPick()*, see page page 80. Figure 43 illustrates this action routine and shows where Pick callbacks are invoked.

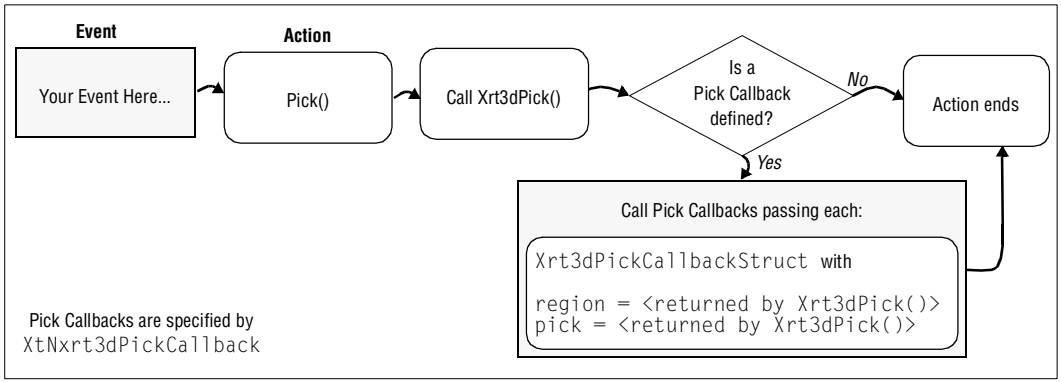


Figure 43 Pick() Action Routine

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

```
*.XtXrt3d.translations: #override \n\
    ~Shift ~Ctrl <Btn1Down>:  Pick()
```

You must use `XtNxrt3dPickCallback` to register a callback to receive the results of the *Xrt3dPick()* 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 */
    Xrt3dRegion     region;     /* Read-only */
    Xrt3dPickResult pick;       /* Read-only */
} Xrt3dPickCallbackStruct;
```

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 *Xrt3dMap()* procedure. You can use the *Map()* action to find out which (x,y) value a user has clicked on. For more information on *Xrt3dMap()*, see page page 82. The way *Map()* works is similar to *Pick()*, illustrated in Figure 43.

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

```
*.XtXrt3d.translations: #override \n\
    ~Shift ~Ctrl <Btn1Down>:  Map()
```

You must use `XtNxrt3dMapCallback` to register a callback to receive the results of the *Xrt3dMap()* 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 */
    Xrt3dRegion     region;     /* Read-only */
    Xrt3dMapResult  map;        /* Read-only */
} Xrt3dMapCallbackStruct;
```

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

7.6.2 Event Handling Procedures

The Xt Intrinsics provide a mechanism for registering interest in events such as mouse movement, mouse button clicks, and keyboard clicks. An XRT/3d program can make use of the event handling mechanisms to react to events that happen over an XRT/3d 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/3d provides procedures for mapping the pixel coordinates of an event to:

- the indices of the point closest to the event coordinates
- the exact graph coordinates at the event coordinates.

Xrt3dPick()

The *Xrt3dPick()* procedure takes an XRT/3d widget handle, (x,y) pixel coordinates and a pointer to an `Xrt3dPickResult` structure. It fills in the `Xrt3dPickResult` structure with information about the point closest to the inputed pixel values (for grid

data, this is the nearest grid point on the surface; for point data, it is the nearest data point in screen coordinate space):

```
Xrt3dRegion
Xrt3dPick(Widget graph, Position pix_x, Position pix_y,
          Xrt3dPickResult *pick)
```

The `Xrt3dPickResult` structure is defined as:

```
typedef struct {
    Position    pix_x, pix_y;
    int         xindex, yindex;
    int         distance;
} Xrt3dPickResult;
```

The fields are broken down as follows:

`pix_x` The x pixel coordinate passed to *Xrt3dPick()*.

`pix_y` The y pixel coordinate passed to *Xrt3dPick()*.

`xindex` For grid data, the index of the X grid line closest to the pixel coordinates. For point data, the series of the point being referenced.

`yindex` For grid data, the index of theY grid line closest to the pixel coordinates. For point data, the point number of the point being referenced.

`distance` The screen distance (in pixels) between the given pixel coordinates and the on-screen display of the data value $Z_{(xindex,\ yindex)}$.

Xrt3dPick() returns one of the following values:

`XRT3D_RGN_NOWHERE` The given pixel coordinates are not close enough to anything to be picked. *xindex* and *yindex* are set to -1.

`XRT3D_RGN_IN_GRAPH` The given pixel coordinates are in the graph area. *xindex* and *yindex* and *distance* are set correctly. For grid data, if a hole in the data is picked, *xindex* and *yindex* are set to -1.

`XRT3D_RGN_IN_LEGEND` The given pixel coordinates are in the legend area. *xindex* and *yindex* are set to -1.

`XRT3D_RGN_IN_HEADER` The given pixel coordinates are not in the graph area, but are in the header area. *xindex* and *yindex* are set to -1.

XRT3D_RGN_IN_FOOTER	The given pixel coordinates are not in the graph area, but are in the footer area. *xindex* and *yindex* are set to -1.

Xrt3dUnpick()

The *Xrt3dUnpick()* procedure is the opposite of *Xrt3dPick()*. It determines the pixel coordinates fo a specified data point. For grid data, it returns the pixel coordinates of the grid point referenced by xindex and yindex. For point data, it returns the pixel coordinates of the scatter plot point specified by xindex (series) and yindex (point number). It is provided for advanced X application developers who use Xlib routines to draw on top of graph images.See Appendix C on page 161 for more details.

```
static void
StartDrag(graph, client_data, bev)
Widget          graph;
caddr_t         client_data;
XButtonEvent    *bev;
{
    /* Start the dragging. If a valid point is selected
     * initialize the drag previewing
     */

    Xrt3dPickResult     p;
    Xrt3dData           *tmp_data

    Xrt3dPick(graph, bev->x, bev->y, &p);
    if (p.xindex != -1 || p.yindex != -1) {
        InitDrag(graph);
        XtVaGetValues(graph, XtNxrt3dSurfaceData, &tmp_data, NULL);
        if (tmp_data->g.type == XRT3D_DATA_POINT) {
            printf("You picked point (%f, %f, %f).\n",
                tmp_data->p.series[p.xindex].points[p.yindex].x,
                tmp_data->p.series[p.xindex].points[p.yindex].y,
                tmp_data->p.series[p.xindex].points[p.yindex].z);
    else {
            printf("You picked point (%d, %d). ", p.xindex, p.yindex);
            printf("The value is %f\n", tmp_data->g.values[p.xindex][p.yindex]);
        }
    }
}
```

Figure 44 Event-handling procedure using Xrt3dPick()

Xrt3dMap()

The *Xrt3dMap()* procedure takes an XRT/3d widget handle, (x,y) pixel coordinates and a pointer to a map structure. For grid data, it maps the center of the pixel point to a point on the surface. Surface values are obtained by interpolating from nearest grid values. For point data, it returns the coordinates of the nearest point (in screen

coordinate space). It writes the pixel coordinates and floating-point graph coordinates into the map structure.

```
Xrt3dRegion
Xrt3dMap(Widget graph, Position pix_x, Position pix_y,
        Xrt3dMapResult *map)
```

The `Xrt3dMapResult` structure is defined as:

```
typedef struct {
    Position    pix_x, pix_y;
    double      x, y, z;
} Xrt3dMapResult;
```

The fields are broken down as follows:

`pix_x` The X coordinate passed into *Xrt3dMap()*.

`pix_y` The Y coordinate passed into *Xrt3dMap()*.

`x` The mapped graph coordinate X-value.

`y` The mapped graph coordinate Y-value.

`z` The mapped graph coordinate Z-value.

Xrt3dMap() returns `XRT3D_RGN_NOWHERE`, `XRT3D_RGN_IN_FOOTER`, `XRT3D_RGN_IN_HEADER`, `XRT3D_RGN_IN_LEGEND` or `XRT3D_RGN_IN_GRAPH`. If the mapped pixel is found to be invalid, the map results's *x*, *y* and *z* elements are all set to `XRT3D_HUGE_VAL`. For grid data, a pixel is invalid if it is not on the surface. For point data, there are no invalid pixels (unless there are no points in the data set, in which case all points are invalid).

Xrt3dUnmap()

The *Xrt3dUnmap()* procedure is the opposite of *Xrt3dMap()*. It maps from graph coordinates to pixel coordinates. It is available for developers of advanced X applications who use Xlib routines to draw on top of graph images. See Appendix C on page 161 for more details.

Xrt3dComputeZValue()

The value of a grid cell or of a data point can be modified programmatically using *Xrt3dComputeZValue()*. The function is passed the indices of the grid cell or the series and point indices of a point and the new pixel location of the point.

This procedure can be used to support the interactive modification of a grid value or a data point. For example, when the user clicks somewhere in the grid area, call *Xrt3dPick()* to determine the closest grid index or point. Then, as the mouse is dragged, new pixel values are passed along with grid or point indices to *Xrt3dComputeZValue()*, which returns the new Z value for the grid cell or point. When the drag has completed, place the new Z value into the appropriate grid cell or point, and assign the updated data structure to the resource `XtNxrt3dSurfaceData`. See the *feedback3d* demo program and source code for an example of this type of user interaction.

Mapping/Picking Bar Charts

Xrt3dMap() and *Xrt3dPick()* can be used on bar charts as well as 3D surface graphs and scatter plots. If passed the coordinates of a pixel that is not part of any bar, *Xrt3dMap()* will return `XRT3D_HUGE_VAL` in the map result's *x*, *y* and *z* elements; *Xrt3dPick()* will return `-1` in the pick result's *x* and *y* indices.

For pixels that are used to display any portion of the top, bottom or sides of a bar, *Xrt3dMap()* returns the actual X, Y and Z values of the pixel.

7.7 Window Resizing

Most applications should allow the user to resize a window containing an XRT/3d widget and have the widget adjust to its new window size.

The XRT/3d widget can be located inside any of the Motif Composite widgets. Usually the Composite widget will determine the appropriate size for the XRT/3d widget and the XRT/3d widget will adjust automatically.

Expose & Resize Callbacks

An application can find out when the XRT/3d widget is exposed or resized by specifying a callback function using the `XtNxrt3dExposeCallback` or `XtNxrt3dResizeCallback` 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 `Xrt3dCallbackStruct` which contains additional information about the callback. See Appendix A on page 133 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/3d widget.

The following code registers a resize callback function:

```
XtAddCallback(mygraph, XtNxrt3dResizeCallback,
    resize_proc, &my_data);
```

The following resize callback causes the legend to disappear when the XRT/3d widget gets too small:

```
void
resize_proc(Widget w, XtPointer client_data, XtPointer call_data)
{
    Dimension   width;
    Boolean     show;

    XtVaGetValues(w,
        XtNwidth,           &width,
        XtNxrt3dLegendShow, &show,
        NULL);

    if ( width <= 300 && show )
        XtVaSetValues(w,
            XtNxrt3dLegendShow, False,
            NULL);
    else if ( width > 300 && !show)
        XtVaSetValues(w,
            XtNxrt3dLegendShow, True,
            NULL);
}
```

8

Advanced XRT/3d Programming

4D Surface Graphs ■ *4D Bar Charts*
Text Objects ■ *Customizing the Distribution Table*
Customizing Legend Labels ■ *Customizing Contour Styles*
Internationalization Support ■ *Porting XRT/3d to Microsoft Windows*
Using C++ with XRT/3d ■ *Using UIL with XRT/3d*
Converting Resource Values to Strings

This section covers topics that programmers of advanced XRT/3d applications will find useful. It assumes that you are already familiar with XRT/3d.

8.1 4D Surface Graphs

For surface and bar charts, XRT/3d can be used to display 4D graphs using color as a fourth dimension. The additional color information is provided to XRT/3d as a second `Xrt3dData` structure using the `XtNxrt3dZoneData` resource.

To create a 4D graph:

- Set DrawZones and DrawShaded to `True`.
- Using the `XtNxrt3dSurfaceData` resource, attach data to provide the 3D surface shape.
- Attach data to be used for deriving the zoning and contouring colors as `XtNxrt3dZoneData`.
- Make sure both `Xrt3dData` structures have the same origin and the same number of X and Y points. If regularly gridded data is being used, then the *xstep* and *ystep* must be identical in both structures. If irregularly-gridded data is being used, the values of *xgrid* and *ygrid* must be identical in both structures. It is not possible to use a regularly gridded data set for the surface data, and an irregularly gridded data for the zone data, or vice versa. If any of these conditions are not met, a 4D graph will not be displayed.

If the zone data has a hole that is not in the surface data, the surface in the region of the hole will be displayed as if the zone data were not attached.

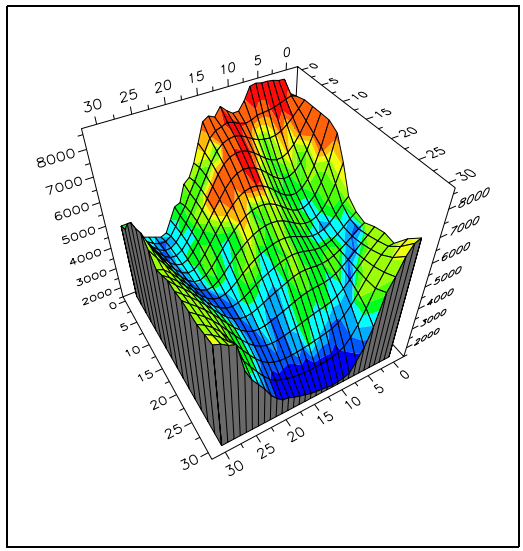


Figure 45 4D graph—Zone/Contour data is different from Surface data

8.2 4D Bar Charts

When `XtNxrt3dZoneData` is supplied for a bar chart, the values in the zone data are used in conjunction with the distribution table to apply zone colors to the bars in the grid.

When zone data is supplied and DrawZones is `True`, the bar is not broken up into separate colored segments. Rather, each bar is individually colored according to the zoned height of the bar. Contours are never drawn when zone data is supplied. Figure 46 gives an example of a 4D bar chart.

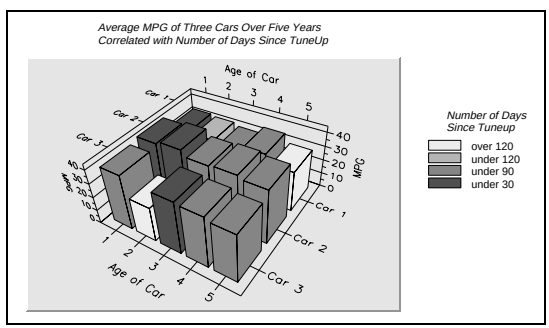


Figure 46 4D Bar Chart

In a bar chart, the zone data structure is only referenced when DrawZones is `True`. A legend is generated based on the distribution table. The legend labels can be replaced, or supplied in a resource with the `XtNxrt3dLegendStrings` resource. Section 8.5 on page 94 has more details on supplying legend strings.

8.3 Text Objects

A surface or bar chart can contain one or more *text objects*. A text object is an independent rectangular region that can be attached to the graph in one of three ways:

- To a particular grid index,
- To a point in 3D space, or
- To a set of pixel coordinates on the widget.

XRT/3d has implemented text objects by defining a new Xt Intrinsics Object class, `xtXrt3dTextObject`. Xt Objects are similar to Motif gadgets, but have no geometry or sensitivity resources. Text objects have their own resources and are manipulated using the same Xt procedures as widgets and gadgets.[1]

You can use text objects in a number of ways, such as annotating points of interest on a surface or as "headers" that appear inside the unit cube.

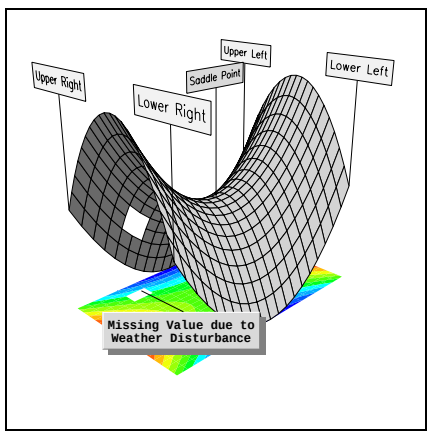


Figure 47 Pixel and Grid Text Objects attached to a surface graph

Text Object Types

The `XtNxrt3dTextAttachMethod` resource specifies how a text object is attached to the graph–this defines the type of text object it is. A *Grid* text object is specified when

1. Because XRT/3d is not subclassed from the Intrinsics Constraint class, you cannot use any Xt function that attempts to manage widgets (such as *XtCreateManagedWidget()* or *XtManageChild()*) on a text object. Attempts to manage text objects cause the program to abort with a fatal error message.

attachment is XRT3D_ATTACH_INDEX or XRT3D_ATTACH_POINT (a Grid text object is also used with point data). A Pixel text object is specified when attachment is XRT3D_ATTACH_PIXEL. A Grid text object displayed on a 2D graph (that is, a surface or scatter plot with DrawMesh and DrawShaded set to False) is referred to as a 2D Grid text object. Figure 48 shows the resources that apply to text objects.

The following points summarize the characteristics of each text object type:

- 2D Grid text objects–attached to the unit cube itself; drawn over the graph; scale and translate with the graph; use stroke fonts.
- 3D Grid text objects–attached to the unit cube itself (either to an X-Y grid point or a point in 3D space); drawn from back to front; rotate, scale, and translate with the graph; use stroke fonts.
- Pixel text objects–similar to a header or footer; attached at a set of widget pixel coordinates; drawn over the graph; stay stationary when the graph rotates, scales or translates; use XmFontList fonts; have a customizable border.

Note: Multiple text objects that intersect are not rendered correctly by XRT/3d. Applications should ensure that text objects are arranged so that they do not intersect each other.

Common Resources

XtNxrt3dTextAdjust	XtNxrt3dTextAttachMethod	XtNxrt3dTextBackgroundColor
XtNxrt3dTextForegroundColor	XtNxrt3dTextLineShow	XtNxrt3dTextOffset[XY] †
XtNxrt3dTextPrintFont	XtNxrt3dTextShow	XtNxrt3dTextStrings
XtNxrt3dTextImage	XtNxrt3dTextImageLayout	XtNxrt3dTextImageMinimumSize

† XtNxrt3dTextOffset[XY] does not apply to 3D Grid text objects

Grid-only Resources

XtNxrt3dTextAttachIndex[XY]	XtNxrt3dTextAttachPoint[XYZ]	XtNxrt3dTextPlane
XtNxrt3dTextStrokeFont	XtNxrt3dTextStrokeSize	XtNxrt3dTextAttachIndexPoint
XtNxrt3dTextAttachIndexSeries	XtNxrt3dTextFont	

Pixel-only Resources

XtNxrt3dTextAttachPixel[XY]	XtNxrt3dTextBorder	XtNxrt3dTextBorderWidth

Figure 48 Text Object Resource Summary

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:

```
Widget   text_obj;
char *strings[] = {"Hello World!", NULL};
text_obj = XtVaCreateWidget("text_obj",
    xtXrt3dTextObjectClass, graph,
    XtNxrt3dTextStrings,    strings,
    ...
    NULL);
```

Text

The text displayed in a text object is specified with the `XtNxrt3dTextStrings` resource. Text objects can display multiple lines of text.

The alignment of multi-line text is specified with the `XtNxrt3dTextAdjust` resource. Text can be left-aligned (`XRT3D_ADJUST_LEFT`), centered (`XRT3D_ADJUST_CENTER`), or right-aligned (`XRT3D_ADJUST_RIGHT`).

3D Grid Object Positioning

To attach a text object to a grid index (for point data see the section on Point Data Object Positioning below), set `XtNxrt3dTextAttachMethod` to `XRT3D_ATTACH_INDEX` and `XtNxrt3dTextAttachIndex[XY]` to an X and Y index on the graph. The X and Y values must be within the X and Y range of the dataset.[1] XRT/3d draws the text object above the maximum Z value on the surface. The offset resources are ignored but the plane the text object is oriented to can be specified using `XtNxrt3dTextPlane`.

To attach a text object to a point in 3D space, set `XtNxrt3dTextAttachMethod` to `XRT3D_ATTACH_POINT` and `XtNxrt3dTextAttachPoint[XYZ]` to an X, Y, and Z point in 3D. These values must be within the data range. If the Z value is less than the Z value on the surface, XRT/3d draws the text object below the minimum Z value; if it is greater or equal to the dataset's Z value (or equal to the hole value), the text object is drawn above the maximum Z value. The offset resources are ignored but the plane the text object is oriented to can be specified using `XtNxrt3dTextPlane`.

The following example positions a 3D Grid text object:

```
XtVaSetValues(my_text,
    XtNxrt3dTextAttachMethod,   XRT3D_ATTACH_POINT,
    XtNxrt3dTextAttachPointX,   Xrt3dDoubleToArgVal(20.0),
    XtNxrt3dTextAttachPointY,   Xrt3dDoubleToArgVal(20.0),
    XtNxrt3dTextAttachPointZ,   Xrt3dDoubleToArgVal(500.0),
    NULL);
```

2D Grid Object Positioning

Text objects on 2D surfaces are attached the same way as 3D surfaces (`XtNxrt3dTextAttachPointZ` is ignored on text objects attached to a point in 2-D space). XRT/3d draws these text objects on top of the graph in the order they were created.

Use the `XtNxrt3dTextOffset[XY]` resources to move the text object away from its point of attachment.

Point Data Object Positioning

To attach a text object to a given series and point, set `XtNxrt3dTextAttachMethod` to `XRT3D_TEXT_ATTACH_INDEX` and set `XtNxrt3dTextAttachIndexSeries` and `XtNxrt3dTextAttachIndexPoint` to the appropriate indices. Text objects can also be attached to any point in 3D space by setting `XtNxrt3dTextAttachMethod` to `XRT3D_TEXT_ATTACH_POINT` in the same way as 3D grid object positioning.

1. If the Z-value at the X and Y index is a hole, no text object is drawn because the hole is considered to be out of data range.

Pixel Object Positioning

To attach a text object to pixel coordinates on the widget, set `XtNxrt3dTextAttachMethod` to `XRT3D_ATTACH_PIXEL` and `XtNxrt3dTextAttachPixel[XY]` to a set of (x,y) pixel coordinates. XRT/3d draws these text objects in the order they were created, after any Grid text objects.

Use the `XtNxrt3dTextOffset[XY]` resources to move the text object away from its point of attachment.

Connecting Line

You can place a line between a text object and its attachment point using the `XtNxrt3dTextLineShow` resource. This resource should be used with the `XtNxrt3dTextOffset[XY]` resources to set a distance between the text object and the attachment point.

Font

Grid text objects use `XtNxrt3dTextStrokeFont` and `XtNxrt3dTextStrokeSize` to specify the font used. See Appendix E on page page 200 for a list of the valid stroke fonts.

Pixel text objects use `XtNxrt3dTextFont` to specify the font used. This resource is specified the same as header, footer, and legend fonts. See section 3.8 on page 33 for further details.

Border

Pixel text objects can be enhanced with a border using the `XtNxrt3dTextBorder` and `XtNxrt3dTextBorderWidth` resources. Any valid XRT/3d border type may be used; see section 3.11 on page 36 for details on XRT/3d borders.

Grid text objects do not use this resource. They have a single-pixel border drawn in the foreground color of the text object. If the background color is `NULL`, no border appears.

Background Image

Text objects can display a background image using the `XtNxrt3dTextImage` and `XtNxrt3dTextImageLayout` resources.

Normally, the minimum size of the text area is determined by the size of the text in that area. Use `XtNxrt3dTextImageMinimumSize` to set the minimum size of the text area to the same size as the background image. (When `True`, a text object is displayed even if there is no text.)

For more details on using images, see section 3.13 on page 38.

Colors

You can set a background and foreground color for a text object using `XtNXrt3dTextBackground` and `XtNxrt3dTextForeground`. For more details on specifying colors, see section 2.7 on page 19.

Text Object Performance

Applications should batch the creation or manipulation of multiple text objects. See section 2.6 on page 19 for more information on batching.

Removing a Text Object

To remove a text object from the display, set `XtNxrt3dTextShow` to `False`. To permanently destroy a text object, call *XtDestroyWidget()*.

8.4 Customizing the Distribution Table

By default, for surface and bar charts, XRT/3d will use a linear distribution table with `XtNxrt3dNumDistnLevels` when DrawContours and/or DrawZones is `True`.

To specify your own distribution table, set `XtNxrt3dDistnMethod` to `XRT3D_DISTN_FROM_TABLE` and then supply your own distribution table using the `XtNxrt3dDistnTable` resource.

`XtNxrt3dDistnTable` specifies a pointer to an `Xrt3dDistnTable` structure:

```
typedef struct {
    int     nentries;
    double  *entry;
} Xrt3dDistnTable;
```

nentries tells XRT/3d how many entries are in the table. *entry* is an array of doubles specifying the boundaries of each level.

When a distribution table is defined, *nentries* defines the number of distribution levels *nlevels*, and `XtNxrt3dNumDistnLevels` is ignored. Recall that to display *nlevels* of distribution, *nlevels* contour line styles are needed, and the fill color from the *(nlevel + 1)*th contour style is needed for the last (highest) zone color when DrawZones is `True`.

For example, the following code defines a distribution table which specifies 2 levels. The first fill color displays all values ≤ -5. The second style displays all values > -5, and ≤ 40. The third style's zone fill color displays all values > 40.

```
static double e[] = { -5.0, 40.0 };
static Xrt3dDistnTable mytable;

mytable.nentries = XtNumber(e);
mytable.entry = e;

XtVaSetValues(mygraph,
    XtNxrt3dDistnMethod, XRT3D_DISTN_FROM_TABLE,
    XtNxrt3dDistnTable,  &mytable,
    NULL);
```

In user-specified distribution tables, the default is to display only the range of the distribution table that is spanned by the data. When `XtNxrt3dLegendDistnRange` is set to `XRT3D_DISTN_RANGE_ALL`, the entire distribution table is shown in the legend.

8.5 Customizing Legend Labels

There are two ways of specifying your own legend labels. You can define your own labeling function, or you can specify labels for each level in the distribution table. Each method is described below.

Legend Label Function

To specify your own legend labels, set `XtNxrt3dLegendLabelFunc` to the name of your own label generating function.

Whenever XRT/3d needs to generate a legend label, it will call your function. Your function must then return a label to use. The form of the function is:

```
char *
MyLegendFunc(Widget graph, int level, double zmin,
             double zmax, char * label)
```

widget is the handle of the XRT/3d widget. *level* is the (zero-indexed) level that a label is being generated for. *zmin* and *zmax* define the level's lowest and highest values. *label* is the string that XRT/3d would generate by default.

To return to default labels, set `XtNxrt3dLegendLabelFunc` to `NULL`.

The *label* string can contain the special characters "`\l`", "`\c`" and "`\r`" for left, center and right field justification. Each field is preceded by one of these special characters. Remember to escape the "`\`" itself with another "`\`". If no special characters are provided, left justification is assumed.

For example, the following code generates normal legend labels, except when the level includes Z=212. It will label this single level as "Boiling Zone":

```
char *
mylabfunc(Widget widget, int level, double zmin,
          double zmax, char * label)
{
    if (zmin < 212.0 && 212.0 <= zmax)
        return("\\cBoiling Zone");
    else
        return(label);
}

XtVaSetValues(graph,
    XtNxrt3dLegendLabelFunc, mylabfunc,
    NULL);
```

If the legend is stepped, and your label function returns `NULL`, XRT/3d will skip the legend entry altogether.

Legend Label Strings

Another way to get custom legend labels is to supply a list of strings to the `XtNxrt3dLegendStrings` resource. For grid data, these strings are used to annotate each level in the distribution table. This method is most useful when displaying a 4D bar chart where groups of bars with the same color require a common label. For point data, these strings are used to annotate each series.

The actual strings used can still be overridden programmatically using the `XtNxrt3dLegendLabelFunc` resource.

8.6 Customizing Contour Styles

By default, for surface and bar charts, XRT/3d provides an array of 100 contour styles. The array used for the graph is specified by the `XtNxrt3dContourStyles` resource. Depending on the type of display, different defaults (color or greyscale) are chosen at runtime. When displaying on monochrome servers, the greyscale styles are used and are automatically dithered to monochrome. You will need to provide custom contour styles if:

- it is important to your application to specify the precise contour style for any particular level;
- if you want to display more than 100 levels; or
- if you want to uniquely identify contour lines. The XRT/3d default contour styles use only black solid lines of width 1.

It is usually easiest to specify more contour styles than will be needed for the number of levels of distribution. If *nstyles* contour styles are provided, and the number of distribution levels is *nlevels*, XRT/3d will calculate the index into the contour styles array for level *i* (*0 ≤ i ≤ nlevels*) as follows[1]:

$$round(i \times (nstyles - 1)/nlevels)$$

Xrt3dContourStyle

Each contour style is defined by an `Xrt3dContourStyle` structure (declared in *<Xm/xrt3d_common.h>*):

```
typedef struct {
    char    *fill_color;        /*for zoning*/
    char    *line_color;        /*line color*/
    int     line_width;         /*line width*/
    Xrt3dLinePattern lpat;      /*2D contours only*/

    /* DONT TOUCH THESE */
    unsigned long   res1;       /* reserved */
    unsigned long   res2;       /* reserved */
} Xrt3dContourStyle;
```

1. defined in code as `#define Round(x) ((int)(x)+.05))`

The Xrt3dContourStyle data structure contains information about how XRT/3d should display contours and zones in a set of data. The fields are broken down as follows:

fill_color — The color used to demarcate the level when DrawZones is True. 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 2.7 on page 19 for details of color specification.

line_color — The color used to demarcate the level's contour line when DrawContours is True. 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 2.7 on page 19 for details of color specification.

line_width — The line width used to demarcate the level's contour line when DrawContours is True. Must be greater than or equal to 0. When line_width is zero, no line is drawn.

lpat — The line pattern used to demarcate the level's contour line when DrawContours is True. Line patterns are only honored when viewing in 2D (i.e. when DrawMesh and DrawShaded are both False and when XtNxrt3dType is XRT3D_TYPE_SURFACE). The line pattern must be one of the XRT3D_LPAT_ constants listed in Figure 49.

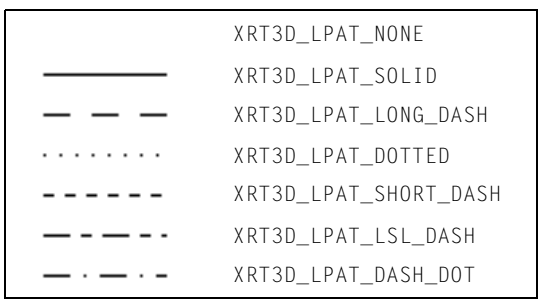


Figure 49 Line Patterns

Explicit Contour Styles

If you do not want to use XRT/3d's default contour styles, you may set your own contour styles.

Contour styles are accessed through the XtNxrt3dContourStyles resource and/or through the *Xrt3dSetNthContourStyle()* and *Xrt3dGetNthContourStyle()* procedures.

XtNxrt3dContourStyles can be used to get or set the entire array of contour styles. When used with *XtGetValues()*, the returned array pointer should be considered read-only. The *Xrt3dDupContourStyles()* procedure may be used to duplicate the returned

contour styles array pointer. For example, to set the first and second contour style's line widths to 3 pixels thick, the following code could be used:

```
Xrt3dContourStyle **cs, **mycs;
XtVaGetValues(myg, XtNxrt3dContourStyles, &cs,
                NULL);
mycs = Xrt3dDupContourStyles(cs);
mycs[0]->line_width = 3;
mycs[1]->line_width = 3;
XtVaSetValues(myg,
    XtNxrt3dContourStyles, mycs,
    NULL);
Xrt3dFreeContourStyles(mycs);
```

A program can get a pointer to a particular `Xrt3dContourStyle` structure using *Xrt3dGetNthContourStyle()*. For example, the following code will print out the 50th contour style's line thickness and double it:

```
Xrt3dContourStyle*cstyle, my_cstyle;
cstyle = Xrt3dGetNthContourStyle(myg, 49,False);
printf("line width for 50th CStyle: %d\n",
    cstyle->line_width);
my_cstyle = * cstyle;
my_cstyle.line_width = my_cstyle.line_width * 2;
Xrt3dSetNthContourStyle(myg, 49,
            &my_cstyle, False);
```

Contour Style Tips

To return to using default contour styles, call the *Xrt3dResetContourStyles()* procedure.

The number of zoning or contouring levels, *nlevels*, is defined by `XtNxrt3dNumDistnLevels`. (If a distribution table has been provided, then *nlevels* is the number of entries in the table.) If DrawContours is `True` and DrawZones is `False`, *nlevels* of contour styles will be needed. If DrawZones is `True`, *nlevels + 1* contour styles will be needed, as the $(nlevels+1)^{th}$ contour style will just be used for defining the last (i.e. highest) zone's fill color.

If you want to have a one-to-one mapping from the supplied contour styles to the styles used on screen, make sure that you supply exactly *nlevels + 1* contour styles.

If too few contour styles are provided for the number of levels requested, XRT/3d will re-sample the supplied contour styles.

8.7 Internationalization Support

ISO Latin1 Character Set

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

2-byte Fonts

2-byte fonts such as Kanji are also supported in the header, footer, and legend areas.

Printing

Xrt3dDrawPS() generates PostScript that uses the full ISO Latin1 character set if it is supported on the PostScript imaging device. This procedure cannot print text that uses 2-byte fonts. Use *Xrt3dOutputPS()* or *Xrt3dOutputXwd()* to print graphs that use 2-byte fonts.

8.8 Porting XRT/3d to Microsoft Windows

XRT/3d provides support for porting Motif XRT/3d 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/3d'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 "3D 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/3d[1]. Using this common API, it is possible to recompile XRT/3d 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/3d and Olectra Chart's 3D Chart control include C++ wrappers.

1. Although this control is also provided as an OCX, only the DLL shares XRT/3d's cross-platform API.

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 3D Chart control is "OlectraChart3D" (32-bit DLL) or "OlectraChart3D16" (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/3d widget as a child of a toplevel shell, Olectra Chart programs use code similar to the following:

```
static HXRT3D graph;
static HWND hwndChart;

hwndChart = GetDlgItem(hWnd, IDCHART);/* defined in .RC file */
graph = Xrt3dCreate();
Xrt3dAttachWindow(graph, hwndChart);/* joins chart and window /*
```

Instead of the single `Widget` handle, Windows programs need a handle for the 3D Chart control (`HXRT3D`) and one for its window on the screen (`HWND`).

Setting and Getting Resources

XRT/3d shares with Olectra Chart an API for setting and getting resource values. *Xrt3dSetValues()* works like *XtVaSetValues()*, allowing you to set one or more resources. *Xrt3dGetValues()* 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 *Xrt3dSetValues()* instead. Note that with you should not call *Xrt3dDoubleToArgVal()* or *Xrt3dArgValToDouble()* before accessing resources of type `double`.

Each XRT/3d resource has a corresponding cross-platform name that can be used in both Motif and 3D Chart code on Windows. Figure 50 lists examples for several resources. For a complete reference, see page page 205.

XRT3D_AXIS_STROKE_FONT	XRT3D_BACKGROUND_COLOR	XRT3D_DRAW_MESH	XRT3D_DRAW_ZONES
XRT3D_FOOTER_BORDER	XRT3D_FOOTER_STRINGS	XRT3D_GRAPH_HEIGHT	XRT3D_HEADER_FONT
XRT3D_LEGEND_SHOW	XRT3D_LEGEND_X	XRT3D_SURFACE_DATA	XRT3D_TYPE
XRT3D_XANNO_METHOD	XRT3D_YDATA_LABELS	XRT3D_ZMAX	XRT3D_ZORIGIN

Figure 50 Some cross-platform resource names

The following example sets three graph properties, assuming the `my_data` variable is defined correctly:

```
Xrt3dSetValues(graph,
    XRT3D_SURFACE_DATA,     my_data,
    XRT3D_TYPE,             XRT3D_TYPE_BAR,
    XRT3D_DRAW_ZONES,       TRUE,
    NULL);
```

Resource Setting Using Strings

The *Xrt3dSetPropString()* and *Xrt3dGetPropString()* 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/3d resources in a resource file (see Appendix G on page 217 for resource file syntax). The following example sets the header background color:

```
Xrt3dSetPropString(graph,
    XRT3D_HEADER_BACKGROUND_COLOR, "orange");
```

Handling Data Types

While most of the data types used in programming XRT/3d and Olectra Chart's 3D Chart control are the same, some differences exist due to differences in the two platforms. XRT/3d and Olectra Chart provides the following cross-platform `typedefs` for these data types that you can use:

Portable typedef	Data Type in XRT/3d	Data Type in Olectra Chart
	`Widget`	`HXRT3D`
XrtBoolean	`Boolean`	`BOOL`
XrtColor	`char *`	`COLORREF`
XrtDimension	`Dimension`	`int`
XrtFont	`XmFontList`	`HFONT`
XrtPosition	`Position`	`int`

8.9 Using C++ with XRT/3d

XRT/3d provides a simple C++ interface to the widget. To use it, include the header file *$XRTHOME/include/Xm/Xrt3d.hxx* in your C++ program. This file creates the three classes described below.

Class	Methods
Xrt3d	■ A constructor that calls *XtCreateXrt3d()*. ■ A destructor that calls *XtDestroyWidget()*. ■ A method for each of the C procedures that take an XRT/3d widget as its first argument. For example, the method for *Xrt3dDrawPS()* 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()*.
Xrt3dDataC	■ A constructor that calls *Xrt3dMakeGridData()*. ■ A constructor that calls *Xrt3dMakeIrGridData()*. ■ A constructor that calls *Xrt3dMakePointData()*. ■ A constructor that calls *Xrt3dMakeDataFromFile()*. ■ A destructor that calls *Xrt3dDestroyData()*. ■ Overloaded = operator to perform *Xrt3dDataCopy()*. ■ Casting support for (`Xrt3dData *`). ■ Methods for each of the internal data structure members.
Xrt3dText	■ A constructor that calls *XtCreateXrt3dText()*. ■ A destructor that calls *XtDestroyWidget()*. ■ A method to set each settable resource. For example, the method to set `XtNxrtTextAdjust` is *setTextAdjust()*. ■ A method to retrieve each resource. For example, the method to retrieve `XtNxrtTextBorder` is *getTextBorder()*.

A simple C++ example is found in *$XRTHOME/src/3d/examples/simple_cpp.cxx*.

8.10 Using UIL with XRT/3d

XRT/3d allows you to use OSF's User Interface Language (UIL) to create XRT/3d widgets. Since UIL does not support any datatypes that are not used by standard Motif widgets, some XRT/3d resources cannot be set using UIL, but can be set in an application with an `MrmNcreateCallback`. See Appendix H on page 223 for a complete list of UIL arguments and their datatypes.

To create a graph with UIL:

- Include the XRT/3d UIL header file *Xrt3d.uil* in your application;
- Call *Xrt3dMrmInitialize()* immediately after calling *MrmInitialize()*.

$XRTHOME/src/3d/examples contains a short UIL example program—*simple_uil.c* and *simple_uil.uil*.

8.11 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 (such as Booleans and integers). XRT/3d automatically registers its built-in resource converters to convert between strings and all of the XRT/3d-specific resource types (such as `XtRXrt3dAdjust` and `XtRXrt3dAlign`).

XRT/3d also provides "reverse converters"—resource converters that convert from XRT/3d-specific resource types to strings. This is useful for writing resource values to a resource file, or saving a widget's state.

To enable the reverse converters, call *Xrt3dSetReverseConverters()*. Every reverse converter allocates the space necessary to store the conversion result. This string must be freed by the calling procedure after use.

The following code fragment shows how these converters could be used to print the value of the legend orientation:

```
void
print_legend_orientation()
{
    Xrt3dAlign    align;
    XrmValue    from, to;

    Xrt3dSetReverseConverters();

    XtVaGetValues(graph,
            XtNxrt3dLegendOrientation, &align, NULL);
    from.addr = (caddr_t) &align;
    from.size = sizeof(align);
    to.addr = NULL;
    if (XtConvertAndStore(graph, XtRXrt3dAlign, &from,
                    XtRString, &to))   {
        printf("Legend Orientation: %s\n", to.addr);
        XtFree(to.addr);
    }
}
```

An alternative, and easier way to do this would be via *Xrt3dGetPropString()*:

```
    void
    print_legend_orientation()
    {
       String       string;
       if (Xrt3dGetPropString(graph, XRT3D_LEGEND_ORIENTATION, &string))
{
            printf("Legend Orientation: %s\n", string);
            Xrt3dFreePropString(string);
       }
    }
```

9

PDS Utility Library

XrtPDSImage ■ Registering Image Types
Using Dynamic or Shared Libraries ■ XrtPDSList Object
Procedures and Methods Reference

9.1 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.

9.1.1 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/3d, 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 9.2.2).

9.2 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
Xrt3dEnableImageType(
    XtPointer   class,
    XtPointer   functions
)
```

where *class* is the widget class (i.e. `xtXrt3dGraphWidgetClass`) 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/3d widget:

```
#include <Xm/Xrt3d.h>
#include <Xm/XrtPDSJpeg.h>
Xrt3dEnableImageType((XtPointer)xtXrt3dGraphWidgetClass,
                  (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.

9.2.1 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.

9.2.2 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:

```
Xrt3dEnableImageType(xtXrt3dGraphWidgetClass, TargaImageUtils)
```

Targa images for all the image resources (such as `XtNxrt3dDataAreaImage`) can now be read. You can also write Targa images by calling *Xrt3dSaveImageGeneric()*:

```
Xrt3dSaveImageGeneric(graph, TARGA_IMAGE_TYPE, "output.tga", NULL)
```

9.2.3 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.

9.3 Using Dynamic or Shared Libraries

The JPEG and PNG image libraries are shipped compiled as static libraries. If you want to create dynamic or shared libraries from them, you have to obtain the source code (freely available on the Internet) and recompile them. Each of the image libraries has a custom XRT module added to it, but is otherwise unmodified. We provide this module in a compiled format ready to be added to a shared library (this module is in *$XRTHOME/lib-shared*).

Please note that this module is only available on the platforms that XRT PDS shared libraries are available on. See the release notes for specific platforms supported.

9.4 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.

9.4.1 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.

9.4.2 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.

9.4.3 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 9.4.6 on page 116.)

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.

9.4.4 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 9.4.6 on page 116 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);
```

9.4.5 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()*.

9.4.6 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 9.4.4 on page 113.

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.

9.4.7 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.

9.5 Procedures and Methods Reference

9.5.1 XrtPDSImage Methods

XrtPDSImageCopy()

Copies the image object.

```
XrtPDSImage
XrtPDSImageCopy(
    XrtPDSImage     image
    )
```

image is an image to be copied. Note that the internal Pixmap(s) are duplicated.

XrtPDSImageCreate()

Creates an XrtPDSImage with the associated Pixmap, Display, width and height.

```
XrtPDSImage
XrtPDSImageCreate(
    Pixmap          pixmap,
    Display         *display,
    int             width,
    int             height
    )
```

pixmap is a pixmap version of an XrtPDSImage; *display* is the X Display for which the image is created; *width* and *height* are the width and height of the XrtPDSImage created.

XrtPDSImageDestroy()

Destroys the image object and any memory associated with it.

```
void
XrtPDSImageDestroy(
    XrtPDSImage     image
    )
```

image is an image to be destroyed. Note that all of the image's internal Pixmaps are freed.

XrtPDSImageGetFilename()

Returns the filename this image was read from. Do not free this memory.

```
String
XrtPDSImageGetFilename(
    XrtPDSImage     image
    )
```

image is an image whose filename is returned.

XrtPDSImageGetWidth()

Returns the width of the image.

```
int
XrtPDSImageGetWidth(
    XrtPDSImage         image
    )
```

image is an image whose width is returned.

XrtPDSImageGetHeight()

Returns the height of the image.

```
int
XrtPDSImageGetHeight(
    XrtPDSImage         image
    )
```

image is an image whose height is returned.

XrtPDSImageGetColorDepth()

Returns the number of bits used to represent each color component for this image (generally 8).

```
int
XrtPDSImageGetColorDepth(
    XrtPDSImage         image
    )
```

image is an image whose color depth is returned.

XrtPDSImageGetClipMask()

Returns the clip mask for this image if it is available or `XmUNSPECIFIED_PIXMAP` if it is not.

```
Pixmap
XrtPDSImageGetClipMask(
    XrtPDSImage         image
    )
```

image is an image whose clip mask is returned.

XrtPDSImageGetImage()

Returns the Pixmap version for this image if it is available or `XmUNSPECIFIED_PIXMAP` if it is not.

```
Pixmap
XrtPDSImageGetImage(
    XrtPDSImage         image
    )
```

image is an image whose Pixmap version is returned.

XrtPDSImageIsSet()

Returns `True` if the Pixmap inside the image object is valid.

```
Boolean
XrtPDSImageIsSet(
    XrtPDSImage         image
    )
```

image is an image that returns `True` if the Pixmap is valid.

XrtPDSImageIsClipMastSet()

Returns `True` if the Pixmap for the clip mask inside the image object is valid.

```
Boolean
XrtPDSImageIsClipMaskSet(
    XrtPDSImage     image
    )
```

image is an image that returns `True` if the Pixmap for the clip mask is valid.

XrtPDSImageIsImage()

Returns True if the image is an XrtPDSImage.

```
Boolean
XrtPDSImageIsImage(
    XrtPDSImage     image
    )
```

image is an image that returns `True` if the image is an XrtPDSImage.

9.5.2 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

XRT/3d Resource Reference

Widget Synopsis
XRT/3d Resources
Inherited Resources

This appendix lists all of the XRT/3d 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/Xrt3d.h>*
Class Name:	XtXrt3d
Class Hierarchy:	Core → XmPrimitive → XtXrt3d
Class Pointer:	xtXrt3dWidgetClass
Instantiation:	`widget = XtCreateWidget(name, xtXrt3dWidgetClass...)`
Functions/Macros:	*XtCreateXrt3d()*, *XtIsXrt3d()*

A.2 XRT/3d Resources

XtNxrt3dAxisStrokeFont **Xrt3dStrokeFont** **XRT3D_SF_ROMAN_SIMPLEX** **CSG**

Specifies the font to be used for the axis annotation. May be set to any of the valid stroke fonts listed in Appendix E on page page 200.

XtNxrt3dAxisStrokeSize **Dimension** **80** **CSG**

Specifies the size of the axis annotation font. The size is measured in thousandths of the unit cube size, and must be between 0 and 1000. See section 2.2 on page 15 for a description of the unit cube.

XtNxrt3dAxisTitleStrokeFont **Xrt3dStrokeFont** **XRT3D_SF_ROMAN_SIMPLEX** **CSG**

Specifies the font to be used for the axis titles. May be set to any of the valid stroke fonts listed in Appendix E on page page 200.

XtNxrt3dAxisTitleStrokeSize **Dimension** **80** **CSG**

Specifies the size of the axis title font. The size is measured in thousandths of the unit cube size, and must be between 0 and 1000. See section 2.2 on page 15 for a description of the unit cube.

XtNxrt3dBackgroundColor **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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dContourStyles **Xrt3dContourStyle ** *dynamic*** **CSG**

For surface and bar charts, the `Xrt3dContourStyle` structure defines a fill color for zoning, and a line color, line width and line style for contouring.

Use `XtNxrt3dContourStyles` to determine what contour styles are currently defined, or to define new styles to be used. To set new contour styles, create a `NULL`-terminated array of contour style structure pointers, and set `XtNxrt3dContourStyles` to be a pointer to this array. When used with *XtGetValues()*, a pointer to an array of `Xrt3dContourStyle` structures is returned.

The number of styles actually used, *nlevels*, depends on `XtNxrt3dDistnMethod` and `XtNxrt3dNumDistnLevels`. If a linear distribution method is being used, then *nlevels* is the value of `XtNxrt3dNumDistnLevels`. If a distribution table is being used, then *nlevels* is the number of table entries.

If more than enough contour styles have been provided, XRT/3d will evenly sample the provided styles. Specifically, if *nstyles* contour styles have been provided, the index of the contour style used at each level *i* is:

round(i x (nstyles - 1)/nlevels)

If not enough contour styles are defined, XRT/3d will re-sample the provided contour styles as needed.

When contour styles are not specified by `XtNxrt3dContourStyles`, XRT/3d will use default contour styles. The default contour styles provided by XRT/3d are determined by the display type, color or greyscale. By default, when displaying on monochrome servers, XRT/3d will use the greyscale contour styles, but will dither the colors in black and white. The default contour styles all have 100 entries. To return to using the default contour styles after having set `XtNxrt3dContourStyles`, call the *Xrt3dResetContourStyles()* procedure.

The procedures *Xrt3dDupContourStyles()* and *Xrt3dFreeContourStyles()*, and the methods *Xrt3dGetNthContourStyle()* and *Xrt3dSetNthContourStyle()* may be used to manipulate contour style structures.

XtNxrt3dDataAreaBackgroundColor **char *** **NULL** **CSG**

Specifies the background color of the sides of the unit cube. When `NULL`, this 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dDataStyles **Xrt3dDataStyles **** ***dynamic*** **CSG**

The `Xrt3dDataStyle` structure defines a line style and a symbol style for a series in a scatter plot. XRT/3d maintains a list of default data styles that a scatter plot can reference. These default data styles may be over-ridden by the user. A side effect of setting this resource is that `XtNxrt3dDataStylesUseDefault` is set to False.

XtNxrt3dDataStylesUseDefault **Boolean** **False** **CSG**

If True, the default data styles are used. If the user sets the `XtNxrt3dDataStyles` resource, this resource is automatically set to False.

XtNxrt3dDebug **Boolean** **False** **CSG**

When `True`, XRT/3d will print application debugging messages to *stderr*.

XtNxrt3dDistnMethod **Xrt3dDistnMethod (enum) XRT3D_DISTN_LINEAR** **CSG**

For surface and bar charts, specifies the type of distribution method used. Must be one of `XRT3D_DISTN_LINEAR` or `XRT3D_DISTN_FROM_TABLE`. Setting this resource to `XRT3D_DISTN_FROM_TABLE` will only have effect if a distribution table has been provided (using `XtNxrt3dDistnTable`).

XtNxrt3dDistnTable **Xrt3dDistnTable *** **NULL** **CSG**

For surface and bar charts, use this resource to specify a custom distribution. The *nentries* element in the `Xrt3dDistnTable` structure specifies the number of entries in the table. *entry* is a pointer to an array of doubles specifying the distribution separation.

```
typedef struct {
    int         nentries;
    double      *entry;
} Xrt3dDistnTable;
```

If `XtNxrt3dDistnMethod` is set to `XRT3D_DISTN_FROM_TABLE`, and a distribution table has been supplied, it will be used to define the distribution. All surface data values less than or equal to the first entry will be graphed in the first zone and displayed using the first contour style. Surface data values greater than the n-1th entry and less than or equal to the nth entry will be graphed in the nth zone. Surface values greater than the last entry will be graphed in the *nentries*+1th zone, and displayed using the last contour style.

Entries that are outside of the graph's Z-range and which are filtered out using `XtNxrt3dLegendLabelFilter` are not displayed in the legend. Entries do not have to be specified in ascending order because XRT/3d will sort the entries before using them. XRT/3d will also cull any duplicate entries.

XtNxrt3dDoubleBuffer **Boolean** **True** **CSG**

When `True`, display updates are first rendered into an off-screen area, then copied to the display area. This reduces flashing on the screen. Setting this resource to `False` will cause graph images to be rendered directly to the display, resulting in less memory utilization by the X window server.

XtNxrt3dDrawContours **Boolean** **False** **CSG**

For surface and bar charts, when `True`, XRT/3d will draw contour lines identifying each entry in the distribution. The contour line color, line width and line pattern are determined by the value of `XtNxrt3dContourStyles`. The distribution is determined by `XtNxrt3dDistnMethod`, `XtNxrt3dDistnTable` and `XtNxrt3dNumDistnLevels`. This resource is ignored when a 4D bar chart is drawn.

If DrawContours is `True` and DrawZones is `False`, XRT/3d will automatically generate a legend entry for each contour line.

This resource is used in conjunction with the DrawMesh, DrawShaded and DrawZones resources to define the 15 basic graph types. See section 3.2 on page 33 for more information.

XtNxrt3dDrawDropLines **Boolean** **False** **CSG**

If `True`, XRT/3d will draw drop lines if the plot type is `XRT3D_TYPE_SCATTER`.

XtNxrt3dDrawHiddenLines **Boolean** **False** **CSG**

For surface and bar charts, this resource determines if mesh, contour and axis lines, hidden from the observer's view, should be drawn. For realistic images, this boolean should be set to `False`. Rendering performance improves when set to `True`.

This resource has no effect when both DrawMesh and DrawContours are `False`, or when either DrawShaded or DrawZones is `True`.

Note: This resource has no effect on scatter plots.

XtNxrt3dDrawMesh **Boolean** **True** **CSG**

When `True`, and when `XtNxrt3dType` is `XRT3D_TYPE_SURFACE`, XRT/3d will display data on a three-dimensional mesh. The mesh lines in either the X or Y direction may be turned off using `XtNxrt3d[XY]MeshShow`. Display of hidden mesh lines is controlled with `XtNxrt3dDrawHiddenLines`. The color of the top and bottom of the mesh is specified with `XtNxrt3dMeshBottomColor` and `XtNxrt3dMeshTopColor`.

When `True`, and when `XtNxrt3dType` is `XRT3D_TYPE_BAR`, XRT/3d will draw the outline of each bar. All bars whose value is above the value of `XtNxrt3dZOrigin` will be outlined using the `XtNxrt3dMeshTopColor`; those bars with value below the value of `XtNxrt3dZOrigin` will be outlined using the color `XtNxrt3dMeshBottomColor`.

Display of hidden lines can be controlled with `XtNxrt3dDrawHiddenLines`.

This resource is used in conjunction with the DrawShaded, DrawContours and DrawZones resources to define the 15 basic graph types. See section 3.2 on page 33 for more information.

Note: This resource has no effect on scatter plots.

XtNxrt3dDrawShaded **Boolean** **False** **CSG**

When `True`, XRT/3d will use flat shading to display the 3D surface (when `XtNxrt3dType` is `XRT3D_TYPE_SURFACE`) and the bar surfaces (when `XtNxrt3dType` is `XRT3D_TYPE_BAR`). The color of the top and bottom of the 3D surface is specified with `XtNxrt3dSurfaceTopColor` and `XtNxrt3dSurfaceBottomColor`. For bar charts, these colors are used to indicate whether a bar is above or below the value of `XtNxrt3dZOrigin`.

This resource is used in conjunction with the DrawMesh, DrawContours and DrawZones resources to define the 15 basic graph types. See section 3.2 on page 33 for more information.

Note: This resource has no effect on scatter plots.

XtNxrt3dDrawZones **Boolean** **False** **CSG**

For surface and bar charts, when `True`, XRT/3d will fill the zones (areas between either contour lines or cells) identifying each entry in the distribution. The zone color is determined by the value of `XtNxrt3dContourStyles`. The distribution is determined by `XtNxrt3dDistnMethod`, `XtNxrt3dDistnTable` and `XtNxrt3dNumDistnLevels`. Normally, `XtNxrt3dSurfaceData` is used to determine zones; however, if `XtNxrt3dZoneData` is defined, it will be used instead, resulting in a 4D display. See sections 8.1 and 8.2 for more information on 4D surfaces and bar charts.

When DrawZones is `True`, XRT/3d will automatically generate a legend entry for each zone color.

This resource is used in conjunction with the DrawMesh, DrawShaded and DrawContours resources to define the 15 basic graph types. See section 3.2 on page 33 for more information.

Note: This resource has no effect on scatter plots.

XtNxrt3dExposeCallback **XtCallbackList** **NULL** **CS**

Specifies the list of callbacks that is called when XRT/3d receives and is finished processing an expose event. These callbacks are called whenever XRT/3d modifies, or redraws the on-

screen image. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget          graph,
    XtPointer       client_data,
    XtPointer       call_data     /* points to an Xrt3dCallbackStruct */
    )
```

where `Xrt3dCallbackStruct` is defined as:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Window      window;
    Region      region;
    int         change;
} Xrt3dCallbackStruct;
```

reason is `XmCR_EXPOSE`. If the widget needed to be repainted as the result of an expose event, then *region* is the area of the widget which needed repainting, otherwise *region* is `NULL`.

change describes which areas of the widget were actually repainted. It is constructed from the following values:

```
#define XRT3D_GRAPH_CHANGED     0x01
#define XRT3D_HEADER_CHANGED    0x02
#define XRT3D_FOOTER_CHANGED    0x04
#define XRT3D_LEGEND_CHANGED    0x08
#define XRT3D_DATA_CHANGED      0x19
#define XRT3D_ALL_CHANGED       0x1F /*All of the above */
```

Note that when the data changes, the graph and legend must also be redrawn.

To avoid infinite loops, if a call to *XtSetValues* on the widget is made from within an expose callback, the list of expose callbacks is not called.

XtNxrt3dFooterAdjust **Xrt3dAdjust (enum)** **XRT3D_ADJUST_CENTER** **CSG**

Specifies how multiple footer text lines should be adjusted within the footer area. Must be one of `XRT3D_ADJUST_CENTER`, `XRT3D_ADJUST_LEFT` or `XRT3D_ADJUST_RIGHT`.

XtNxrt3dFooterBackgroundColor **char *** **NULL** **CSG**

Specifies the footer area background color. When `NULL` and `XtNxrt3dFooterImage` is `NULL`, the footer 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dFooterBorder **Xrt3dBorder (enum)** **XRT3D_BORDER_NONE** **CSG**

Specifies the style of border used to identify the footer area. Valid styles are `XRT3D_BORDER_NONE`, `XRT3D_BORDER_PLAIN`, `XRT3D_BORDER_3D_IN`, `XRT3D_BORDER_3D_OUT`, `XRT3D_BORDER_ETCHED_IN`, `XRT3D_BORDER_ETCHED_OUT`, `XRT3D_BORDER_FRAME_IN`, `XRT3D_BORDER_FRAME_OUT`, `XRT3D_BORDER_BEVEL`, and `XRT3D_BORDER_SHADOW`.

XtNxrt3dFooterBorderWidth **Dimension** **2** **CSG**

Specifies the width of the footer area border in pixels. Must be between 0 and 20.

XtNxrt3dFooterFont **XmFontList** **"fixed"** **CSG**

Specifies the font to use for the footer strings.

XtNxrt3dFooterForegroundColor **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 2.7 on page 19 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dFooterImage **Xrt3dImage** **NULL** **CSG**

Specifies an image to display in the background of the footer area. See Background Images on page 38 for more information.

XtNxrt3dFooterImageLayout **Xrt3dImageLayout** **XRT3D_IMAGELAYOUT_CENTERED** **CSG**

Specifies the layout of the background image. Valid values are: `XRT3D_IMAGELAYOUT_CENTERED` and `XRT3D_IMAGELAYOUT_TILED`.

XtNxrt3dFooterImageMinimumSize **Boolean** **False** **CSG**

Specifies that the minimum size of the footer area should be determined by the size of the background image.

XtNxrt3dFooterStrings **char **** **NULL** **CSG**

Specifies a `NULL`-terminated list of strings to be displayed in the footer area.

XtNxrt3dFooterWidth **Dimension** ***dynamic*** **G**
XtNxrt3dFooterHeight **Dimension** ***dynamic*** **G**

Contains the height and width of the footer area in pixels.

XtNxrt3dFooterX **Position** **centered under graph area** **CSG**
XtNxrt3dFooterY **Position** **centered under graph area** **CSG**

Specifies the x or y pixel offset of the top left corner of the footer, relative to the top left corner of the window. For example, (10, 70) will cause the footer to begin 10 pixels to the right of and 70 pixels down from the top left corner of the window. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrt3dFooterXUseDefault **Boolean** **True** **CSG**
XtNxrt3dFooterYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrt3dFooterX` and `XtNxrt3dFooterY` values are determined by XRT/3d at render-time. When explicit `XtNxrt3dFooterX` and `XtNxrt3dFooterY` values are provided, XRT/3d sets these Booleans to `False`.

XtNxrt3dForegroundColor **char *** **"Black"** **CSG**

Specifies the window foreground color which is used for header, footer and legend text. 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dGraphBackgroundColor **char *** **NULL** **CSG**

Specifies the graph area background color. When `XtNxrt3dGraphBackgroundColor` and `XtNxrt3dGraphImage` 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dGraphBorder **Xrt3dBorder (enum)** **XRT3D_BORDER_NONE** **CSG**

Specifies the style of border used to identify the graph area. Valid styles are `XRT3D_BORDER_NONE`, `XRT3D_BORDER_PLAIN`, `XRT3D_BORDER_3D_IN`, `XRT3D_BORDER_3D_OUT`, `XRT3D_BORDER_ETCHED_IN`, `XRT3D_BORDER_ETCHED_OUT`, `XRT3D_BORDER_FRAME_IN`, `XRT3D_BORDER_FRAME_OUT`, `XRT3D_BORDER_BEVEL`, and `XRT3D_BORDER_SHADOW`.

XtNxrt3dGraphBorderWidth **Dimension** **2** **CSG**

Specifies the width of the graph area border in pixels. Must be between 0 and 20.

XtNxrt3dGraphForegroundColor **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 2.7 on page 19 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dGraphImage **Xrt3dImage** **NULL** **CSG**

Specifies an image to display in the background of the graph area. See Background Images on page 38 for more information.

XtNxrt3dGraphImageLayout **Xrt3dImageLayout** **XRT3D_IMAGELAYOUT_CENTERED** **CSG**

Specifies the layout of the background image. Valid values are: `XRT3D_IMAGELAYOUT_CENTERED` and `XRT3D_IMAGELAYOUT_TILED`.

XtNxrt3dGraphWidth **Dimension** ***dynamic*** **CSG**
XtNxrt3dGraphHeight **Dimension** ***dynamic*** **CSG**

Specifies the height and width of the graph area in pixels. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrt3dGraphWidthUseDefault **Boolean** **True** **CSG**
XtNxrt3dGraphHeightUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrt3dGraphWidth` and `XtNxrt3dGraphHeight` values are determined by XRT/3d at render-time. When explicit `XtNxrt3dGraphWidth` and `XtNxrt3dGraphHeight` values are provided, XRT/3d sets these Booleans to `False`.

XtNxrt3dGraphX **Position** **derived from data** **CSG**
XtNxrt3dGraphY **Position** **derived from data** **CSG**

Specifies the x or y pixel offset of the top left corner of the graph area, relative to the top left corner of the window. For example, (10, 5) will cause the graph area to begin 10 pixels to the right, and 5 pixels down from the top left corner of the window. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrt3dGraphXUseDefault **Boolean** **True** **CSG**
XtNxrt3dGraphYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrt3dGraphX` and `XtNxrt3dGraphY` values are determined by XRT/3d at render-time. When explicit `XtNxrt3dGraphX` and `XtNxrt3dGraphY` values are provided, XRT/3d sets these Booleans to `False`. You cannot set these resources to `False` unless you have previously provided `XtNxrt3dGraphX` or `XtNxrt3dGraphY` values.

XtNxrt3dHeaderAdjust **Xrt3dAdjust (enum)** **XRT3D_ADJUST_CENTER** **CSG**

Specifies how multiple header text lines should be adjusted within the header area. Must be one of `XRT3D_ADJUST_CENTER`, `XRT3D_ADJUST_LEFT` or `XRT3D_ADJUST_RIGHT`.

XtNxrt3dHeaderBackgroundColor **char *** **NULL** **CSG**

Specifies the header area background color. When `XtNxrt3dHeaderBackgroundColor` and `XtNxrt3dHeaderImage` 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dHeaderBorder **Xrt3dBorder (enum)** **XRT3D_BORDER_NONE** **CSG**

Specifies the style of border used to identify the header area. Valid styles are `XRT3D_BORDER_NONE`, `XRT3D_BORDER_PLAIN`, `XRT3D_BORDER_3D_IN`, `XRT3D_BORDER_3D_OUT`, `XRT3D_BORDER_ETCHED_IN`, `XRT3D_BORDER_ETCHED_OUT`, `XRT3D_BORDER_FRAME_IN`, `XRT3D_BORDER_FRAME_OUT`, `XRT3D_BORDER_BEVEL`, and `XRT3D_BORDER_SHADOW`.

XtNxrt3dHeaderBorderWidth **Dimension** **2** **CSG**

Specifies the width of the header area border in pixels. Must be between 0 and 20.

XtNxrt3dHeaderFont **XmFontList** **"fixed"** **CSG**

Specifies the font to use for the header strings.

XtNxrt3dHeaderForegroundColor **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 2.7 on page 19 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dHeaderImage **Xrt3dImage** **NULL** **CSG**

Specifies an image to display in the background of the header area. See Background Images on page 38 for more information.

XtNxrt3dHeaderImageLayout **Xrt3dImageLayout** **XRT3D_IMAGELAYOUT_CENTERED** **CSG**

Specifies the layout of the background image. Valid values are: `XRT3D_IMAGELAYOUT_CENTERED`and `XRT3D_IMAGELAYOUT_TILED`.

XtNxrt3dHeaderImageMinimumSize **Boolean** **False** **CSG**

Specifies that the minimum size of the header area should be determined by the size of the background image.

XtNxrt3dHeaderStrings **char **** **NULL** **CSG**

Specifies a `NULL`-terminated list of strings to be displayed in the header area.

XtNxrt3dHeaderWidth **Dimension** ***dynamic*** **G**
XtNxrt3dHeaderHeight **Dimension** ***dynamic*** **G**

Contains the height and width of the header area in pixels.

XtNxrt3dHeaderX **Position** **centered above graph area** **CSG**
XtNxrt3dHeaderY **Position** **centered above graph area** **CSG**

Specifies the x or y pixel offset of the top left corner of the header, relative to the top left corner of the window. For example, (10, 20) will cause the header to begin 10 pixels to the right of, and 20 pixels down from, the top left corner of the window. A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrt3dHeaderXUseDefault **Boolean** **True** **CSG**
XtNxrt3dHeaderYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrt3dHeaderX` and `XtNxrt3dHeaderY` values are determined by XRT/3d at render-time. When explicit `XtNxrt3dHeaderX` and `XtNxrt3dHeaderY` values are provided, XRT/3d sets these Booleans to `False`.

XtNxrt3dImage **Xrt3dImage** **NULL** **CSG**

Specifies an image to display in the background of the widget. See Background Images on page 38 for more information.

XtNxrt3dImageLayout **Xrt3dImageLayout** **XRT3D_IMAGELAYOUT_CENTERED** **CSG**

Specifies the layout of the background image. Valid values are: XRT3D_IMAGELAYOUT_CENTERED and XRT3D_IMAGELAYOUT_TILED.

XtNxrt3dLegendAnchor **Xrt3dAnchor (enum)** **XRT3D_ANCHOR_EAST** **CSG**

Specifies where to anchor the legend to the window. Valid values are: XRT3D_ANCHOR_NORTH, XRT3D_ANCHOR_SOUTH, XRT3D_ANCHOR_EAST, XRT3D_ANCHOR_WEST, XRT3D_ANCHOR_NORTHWEST, XRT3D_ANCHOR_NORTHEAST, XRT3D_ANCHOR_SOUTHWEST, XRT3D_ANCHOR_SOUTHEAST.

XtNxrt3dLegendBackgroundColor **char *** **NULL** **CSG**

Specifies the legend area background color. When XtNxrt3dLegendBackgroundColor and XtNxrt3dLegendImage are both NULL, the legend 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dLegendBorder **Xrt3dBorder (enum)** **XRT3D_BORDER_NONE** **CSG**

Specifies the style of border used to identify the legend area. Valid styles are XRT3D_BORDER_NONE, XRT3D_BORDER_PLAIN, XRT3D_BORDER_3D_IN, XRT3D_BORDER_3D_OUT, XRT3D_BORDER_ETCHED_IN, XRT3D_BORDER_ETCHED_OUT, XRT3D_BORDER_FRAME_IN, XRT3D_BORDER_FRAME_OUT, XRT3D_BORDER_BEVEL, and XRT3D_BORDER_SHADOW.

XtNxrt3dLegendBorderWidth **Dimension** **2** **CSG**

Specifies the width of the legend area border in pixels. Must be between 0 and 20.

XtNxrt3dLegendDistnRange **Xrt3dDistnRange** **XRT3D_DISTN_RANGE_DATA** **CSG**

Given a user-specified distribution table, this resource specifies whether the range of the distribution is spanned by the data (the default) or shows the range specified by the entire distribution table.

XtNxrt3dLegendFont **XmFontList** **"fixed"** **CSG**

Specifies the font to be used in the legend.

XtNxrt3dLegendForegroundColor **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 2.7 on page 19 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dLegendImage **Xrt3dImage** **NULL** **CSG**

Specifies an image to display in the background of the legend. See Background Images on page 38 for more information.

XtNxrt3dLegendImageLayout **Xrt3dImageLayout** **XRT3D_IMAGELAYOUT_CENTERED** **CSG**

Specifies the layout of the background image. Valid values are: `XRT3D_IMAGELAYOUT_CENTERED`and `XRT3D_IMAGELAYOUT_TILED`.

XtNxrt3dLegendLabelFilter **int** **1** **CSG**

When this resource is set to be *n*, every n^{th} legend glyph/label combination will be drawn.

XtNxrt3dLegendLabelFunc **XtFunction** **NULL** **CSG**

This resource can be used to provide legend labels. If not `NULL`, XRT/3d will call the supplied function to obtain a label for each legend level (starting from 0). The form of the function is:

```
char *
MyLegendFunction(
    Widget      graph;
    int         level;
    double      zmin, zmax;
    char *      *label;
    )
```

Once a specific function is provided, XRT/3d will call it, passing as parameters the *widget* handle, the *level* for which a label is required (zero-indexed), the *zmin* and *zmax* that define the level, and the *label* that XRT/3d would normally calculate by default. This function should return a string that XRT/3d will use for that level's legend label. If the legend is stepped, and this function returns `NULL`, XRT/3d will skip the legend entry.

The label string can contain up to thirty fields which are delimited by '`\l`', '`\c`' and '`\r`' for *left*, *center* and *right* justification. For example, "`\r32.0\cto\r39.6`". Remember to escape the '`\`' with another '`\`'.

XtNxrt3dLegendOrientation **Xrt3dAlign (enum)** **XRT3D_ALIGN_VERTICAL** **CSG**

Specifies the orientation of the legend. Valid values are `XRT3D_ALIGN_HORIZONTAL` and `XRT3D_ALIGN_VERTICAL`.

XtNxrt3dLegendShow **Boolean** **True** **CSG**

Determines if the legend should be displayed. Legends are created automatically by XRT/3d whenever contours or zones are drawn.

XtNxrt3dLegendStrings **char **** **NULL** **CSG**

Specifies strings to be used in the legend. For surface and bar charts, the number of distribution levels is determined by the value of `XtNxrt3dNumDistnLevels` or `XtNxrt3dDistnTable`, and the setting of `XtNxrt3dLegendStyle`. For scatter plots there is one legend entry per series. If too few strings are provided, the corresponding entries in the legend are dropped. A legend is generated only when either contours or zones are drawn, and `XtNxrt3dLegendShow` is `True`.

These legend strings override anything that the widget would normally have generated; however, if an `XtNxrt3dLegendLabelFunc` has been specified, the legend strings may be overridden dynamically at runtime. This resource is most often used to annotate particular groups of bars in a 4D bar chart.

XtNxrt3dLegendStyle **Xrt3dLegendStyle** **XRT3D_LEGEND_STYLE_CONTINUOUS** **CSG**

For surface and bar charts, determines the legend style and must be one of `XRT3D_LEGEND_STYLE_STEPPED` or `XRT3D_LEGEND_STYLE_CONTINUOUS`. Stepped legends have the range printed for each zone. Continuous legends have the level printed at the division between two adjacent zones, and are more compact.

XtNxrt3dLegendTitle **char *** **NULL** **CSG**

Allows the user to specify the legend title. The title will be limited to a single line, centered above the legend and using the legend font.

XtNxrt3dLegendWidth **Dimension** ***dynamic*** **G**
XtNxrt3dLegendHeight **Dimension** ***dynamic*** **G**

Contains the height and width of the legend area in pixels.

XtNxrt3dLegendX **Position** **depends on anchor** **CSG**
XtNxrt3dLegendY **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`.

XtNxrt3dLegendXUseDefault **Boolean** **True** **CSG**
XtNxrt3dLegendYUseDefault **Boolean** **True** **CSG**

When `True`, `XtNxrt3dLegendX` and `XtNxrt3dLegendY` values are determined by XRT/3d at render-time. When explicit `XtNxrt3dLegendX` and `XtNxrt3dLegendY` values are provided, XRT/3d sets these Booleans to `False`.

XtNxrt3dMapCallback **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 *Xrt3dMap()* procedure. A pointer to an `Xrt3dMapCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget          graph,
    XtPointer       client_data,
    XtPointer       call_data     /* points to an Xrt3dMapCallbackStruct */
    )
```

1. No default translation is bound to this action.

where XrtMapCallbackStruct is defined as:

```
typedef struct {
    int                 reason;     /* Read-only */
    XEvent              *event;     /* Read-only */
    String              *params;    /* Read-only */
    Cardinal            nparams;    /* Read-only */
    Xrt3dRegion         region;     /* Read-only */
    Xrt3dMapResult      map;        /* Read-only */
} Xrt3dMapCallbackStruct;
```

reason is always XRT3D_REASON_MAP; *region* is the area of the widget in which the event occurred; *map* contains the graph coordinates calculated by *Xrt3dMap()*.

XtNxrt3dMeshBottomColor **char *** **"Black"** **CSG**
XtNxrt3dMeshTopColor **char *** **"Black"** **CSG**

Specifies the color of the 3D surface mesh lines or bar chart outlines displayed when XtNxrt3dDrawMesh is True. 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 2.7 on page 19 for details of color specification.

For bar charts, the top color is used for bars with a value greater than XtNxrt3dZOrigin, and the bottom color is used for bars with a value less than XtNxrt3dZOrigin.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dModifyCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when starting and finishing user interaction (scale, zoom, translate, rotate, or reset). Modify callbacks are triggered by the *ModifyStart()* and *ModifyEnd()* actions. A pointer to an Xrt3dModifyCallbackStruct is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget          graph,
    XtPointer       client_data,
    XtPointer       call_data  /* points to an Xrt3dModifyCallbackStruct */
    )
```

where Xrt3dModifyCallbackStruct is defined as:

```
typedef struct {
    int             reason;     /* Read-only */
    XEvent          *event;     /* Read-only */
    String          *params;    /* Read-only */
    Cardinal        nparams;    /* Read-only */
    Boolean         doit;
} Xrt3dModifyCallbackStruct;
```

reason is either XRT3D_REASON_MODIFY_START or XRT3D_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 XRT3D_REASON_MODIFY_START.

XtNxrt3dNumDistnLevels **int** **10** **CSG**

For surface and bar charts, specifies the number of distribution levels (i.e. *nlevels*) used when DrawContours and/or DrawZones is `True`. Any value between 0 and 100 is valid. Setting it to 0 causes no contour lines to be drawn, and if DrawZones is `True`, only one zone is drawn. If a custom distribution table has been provided, and `XtNxrt3dDistnMethod` is `XRT3D_DISTN_FROM_TABLE`, the number of distribution levels is the number of entries in the table and `XtNxrt3dNumDistnLevels` is ignored.

XtNxrt3dPerspectiveDepth **double** **2.5** **CSG**

Controls the perspective effect. Range is 1 to `XRT3D_HUGE_VAL`. Small values exaggerate perspective while large values reduce it. To reduce the perspective effect and approach a parallel projection, set this resource to a very high number (such as 100,000).

XtNxrt3dPickCallback **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 *Xrt3dPick()* procedure. A pointer to an `Xrt3dPickCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget          graph,
    XtPointer       client_data,
    XtPointer       call_data    /* points to an Xrt3dPickCallbackStruct */
    )
```

where `Xrt3dPickCallbackStruct` is defined as:

```
typedef struct {
    int                 reason;     /* Read-only */
    XEvent              *event;     /* Read-only */
    String              *params;    /* Read-only */
    Cardinal            nparams;    /* Read-only */
    Xrt3dRegion         region;     /* Read-only */
    Xrt3dPickResult     pick;       /* Read-only */
} Xrt3dPickCallbackStruct;
```

reason is always `XRT3D_REASON_PICK`; *region* is the area of the widget in which the event occurred; *pick* contains the grid coordinates calculated by *Xrt3dPick()*.

XtNxrt3dPreviewMethod **Xrt3dPreviewMethod** **XRT3D_PREVIEW_CUBE** **CSG**

Controls how interactive rotations are previewed to the user. `XRT3D_PREVIEW_CUBE` draws only the outline of the cube. `XRT3D_PREVIEW_FULL` draws the entire chart during rotations (useful for small charts or bar charts).

1. No default translation is bound to this action.

XtNxrt3dProjectZMax **int** **0** **CSG**
XtNxrt3dProjectZMin **int** **0** **CSG**

For surface and bar charts, specifies whether to project contours and/or zones on the *z=zmin* or *z=zmax* plane. These resources are specified using a combination of the following constants:

```
#define XRT3D_PROJECT_CONTOURS  0x1
#define XRT3D_PROJECT_ZONES     0x2
```

Contours/zones are projected regardless of the setting of DrawContours and DrawZones. These resources are ignored in a 2D graph, and when `XtNxrt3dType` is `XRT3D_TYPE_BAR`.

For example, to project both contours and zones, set the resource to:

```
XRT3D_PROJECT_CONTOURS | XRT3D_PROJECT_ZONES
```

XtNxrt3dRepaint **Boolean** **True** **CSG**

When this resource is `True`, any changes made to the widget are rendered immediately. In order to batch changes to the widget, set this resource to `False`, make the changes, and then on the last *XtSetValues()* call, set this resource to `True` again.

XtNxrt3dResizeCallback **XtCallbackList** **NULL** **CS**

Specifies the list of callbacks that is called when XRT/3d receives and is finished processing a resize event. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget          graph,
    XtPointer       client_data,
    XtPointer       call_data   /* points to an Xrt3dCallbackStruct */
    )
```

where `Xrt3dCallbackStruct` is defined as:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Window      window;
    Region      region;
    int         change;
} Xrt3dCallbackStruct;
```

reason is `XmCR_RESIZE`. For resize callbacks, the event is always `NULL`.

region and *change* are not used in the resize callback.

XtNxrt3dRotateCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when the user is rotating a 3D graph. Rotate callbacks are triggered by the *RotateUpdate()* action. A pointer to an `Xrt3dRotateCallbackStruct` is passed to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget          graph,
    XtPointer       client_data,
    XtPointer       call_data /* points to an Xrt3dRotateCallbackStruct */
    )
```

where Xrt3dRotateCallbackStruct is defined as:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    String      *params;    /* Read-only */
    Cardinal    nparams;    /* Read-only */
    double      xrotation;
    double      yrotation;
    double      zrotation;
    Boolean     doit;
} Xrt3dRotateCallbackStruct;
```

reason is always XRT3D_REASON_ROTATE; *xrotation, yrotation* and *zrotation* are the proposed new values for the resources XtNxrt3d[XYZ]Rotation; *doit* is a flag (default True) that determines whether the rotation should be allowed. To disallow rotation, set *doit* to False; to modify the proposed resources values, change the values in the structure.

XtNxrt3dSolidSurface **Boolean** **False** **CSG**

For surface and bar charts, when this resource is True, and DrawMesh or DrawShaded is True, XRT/3d will render the 3D surface as a solid object by adding "sides" to the surface. The sides drop from the surface to the minimum Z value. The mesh and surface bottom colors are used when drawing the "sides". This resource has no effect on bar charts.

XtNxrt3dSurfaceBottomColor **char *** **"dim grey"** **CSG**
XtNxrt3dSurfaceTopColor **char *** **"light grey"** **CSG**

Specifies the color of the bottom and top of the 3D surface and bar chart facets displayed when XtNxrt3dDrawShaded is True. 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 2.7 on page 19 for details of color specification.

For bar charts, the top color is used for bars with a value greater than XtNxrt3dZOrigin, and the bottom color is used for bars with a value less than XtNxrt3dZOrigin.

XtNxrt3dSurfaceData **Xrt3dData *** **NULL** **CSG**

Specifies the data to be displayed in 3D surface graphs, bar charts and scatter plots. The Xrt3dData structure should be allocated using the procedure *Xrt3dMakeGridData()* or *Xrt3dMakeDataFromFile()*. See section 6.2 on page 66 for information on this structure.

XtNxrt3dTextList **WidgetList** **NULL** **CSG**

Specifies a NULL-terminated list of this widget's text objects. The value of this resource is automatically updated when a new text object is created. A list of text objects can be specified in a resource file; XRT/3d creates a new text object for each name in the list using *Xrt3dTextCreate()*.

XtNxrt3dTransformCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when the user is translating, scaling or zooming a graph. Transform callbacks are triggered by the *TranslateUpdate()*, *ZoomEnd()*, and *ViewReset()* actions. A pointer to an Xrt3dTransformCallbackStruct is passed

to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget      graph,
    XtPointer   client_data,
    XtPointer   call_data /* points to an Xrt3dTransformCallbackStruct */
    )
```

where `Xrt3dTransformCallbackStruct` is defined as:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    String      *params;    /* Read-only */
    Cardinal    nparams;    /* Read-only */
    double      scale;
    double      xtranslate;
    double      ytranslate;
    Boolean     doit;
} Xrt3dTransformCallbackStruct;
```

reason is always `XRT3D_REASON_TRANSFORM`; *scale, xtranslate* and *ytranslate* are the proposed new values for the resources `XtNxrt3dViewScale` and `XtNxrt3dView[XY]Translate`; *doit* is a flag (default `True`) that determines whether the transformation should be allowed. To disallow all transformations, set *doit* to `False`; to modify the proposed resources values, change the values in the structure.

XtNxrt3dType **Xrt3dType** **XRT3D_TYPE_SURFACE** **CSG**

Specifies the basic type of graph that will be drawn. It may be `XRT3D_TYPE_SURFACE`, `XRT3D_TYPE_BAR` or `XRT3D_TYPE_SCATTER`. For surface and bar charts, many variations can be displayed by manipulating the four major Boolean resources. See section 3.2 on page 33 for more information.

XtNxrt3dTextList **WidgetList** **NULL** **CSG**

Specifies a `NULL`-terminated list of this widget's text objects. The value of this resource is automatically updated when a new text object is created. A list of text objects can be specified in a resource file; XRT/3d creates a new text object for each name in the list using *Xrt3dTextCreate()*.

XtNxrt3dTransformCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that are called to notify the application when the user is translating, scaling or zooming a graph. Transform callbacks are triggered by the *TranslateUpdate()*, *ZoomEnd()*, and *ViewReset()* actions. A pointer to an `Xrt3dTransformCallbackStruct` is passed

to each callback. Add callbacks to the list using the *XtAddCallback()* procedure. The form of the callback is:

```
void
CallbackProcedure(
    Widget      graph,
    XtPointer   client_data,
    XtPointer   call_data /* points to an Xrt3dTransformCallbackStruct */
    )
```

where `Xrt3dTransformCallbackStruct` is defined as:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    String      *params;    /* Read-only */
    Cardinal    nparams;    /* Read-only */
    double      scale;
    double      xtranslate;
    double      ytranslate;
    Boolean     doit;
} Xrt3dTransformCallbackStruct;
```

reason is always `XRT3D_REASON_TRANSFORM`; *scale, xtranslate* and *ytranslate* are the proposed new values for the resources `XtNxrt3dViewScale` and `XtNxrt3dView[XY]Translate`; *doit* is a flag (default `True`) that determines whether the transformation should be allowed. To disallow all transformations, set *doit* to `False`; to modify the proposed resources values, change the values in the structure.

XtNxrt3dViewNormalized **Boolean** **True** **CSG**

When `True`, XRT/3d always calculates the full view (before taking into account any view scaling or translation) so as to maximize the view without causing any portion of the view to be clipped. Dynamically rotating when this resource is `True` will look "jerky". To eliminate "jerky rotation", set this resource to `False`.

XtNxrt3dViewScale **double** **1.0** **CSG**

Specifies the degree of scaling (i.e. zooming) up or down from a full view (i.e. a maximal, non-clipped view). Valid values must be between 1/30 and 30.0. This resource is updated if the user dynamically scales or zooms their view. See section 7.4.1 on page 74 for information on dynamic scaling.

XtNxrt3dViewXTranslate **double** **0.0** **CSG**
XtNxrt3dViewYTranslate **double** **0.0** **CSG**

Specifies the degree of view translation from a full view (i.e. a maximal, non-clipped view). The translations are relative to a normalized box, with origin in the bottom left corner. Translation of more than a full unit (i.e. more than ± 1.0) will cause the view to translate "out of view". This resource is updated if the user dynamically translates their view. See section 7.4.1 on page 74 for information on dynamic translation.

XtNxrt3dXAnnoMethod	**Xrt3dAnnoMethod**	**XRT3D_ANNO_VALUES**	**CSG**
XtNxrt3dYAnnoMethod	**Xrt3dAnnoMethod**	**XRT3D_ANNO_VALUES**	**CSG**
XtNxrt3dZAnnoMethod	**Xrt3dAnnoMethod**	**XRT3D_ANNO_VALUES**	**CSG**

Specifies the method used to annotate the axis. It is an enumerated type, with possible values `XRT3D_ANNO_VALUES`, `XRT3D_ANNO_DATA_LABELS`, and `XRT3D_ANNO_VALUE_LABELS`. If `XRT3D_ANNO_VALUES`, the strings are generated by the widget automatically, based on the range of the data. If `XRT3D_ANNO_DATA_LABELS`, the value of the resource `XtNxrt3d[XY]DataLabels` is used; this method is not available for the Z-axis. If `XRT3D_ANNO_VALUE_LABELS`, the value of the resource `XtNxrt3d[XYZ]ValueLabels` is used.

Note that data labels are not used in scatter plots.

XtNxrt3dXAxisShow	**Boolean**	**True**	**CSG**
XtNxrt3dYAxisShow	**Boolean**	**True**	**CSG**
XtNxrt3dZAxisShow	**Boolean**	**True**	**CSG**

Determines if the X-, Y-, or Z-axis should be displayed.

XtNxrt3dXAxisTitle	**char ***	**NULL**	**CSG**
XtNxrt3dYAxisTitle	**char ***	**NULL**	**CSG**
XtNxrt3dZAxisTitle	**char ***	**NULL**	**CSG**

Specifies the title to display on the X-, Y- and Z-axis.

XtNxrt3dXBarFormat	**Xrt3dBarFormat**	**XRT3D_BAR_FIXED**	**CSG**
XtNxrt3dYBarFormat	**Xrt3dBarFormat**	**XRT3D_BAR_FIXED**	**CSG**

Specifies the representation format of the bar chart along each of the gridded axes. When set to `XRT3D_BAR_FIXED`, the bars in the chart are centered about their X values and Y values, and each bar is a fixed width. When set to `XRT3D_BAR_HISTOGRAM`, each bar fills the space up to its adjoining bar, and the width of each bar is derived from the separation between adjacent lines in the grid.

XtNxrt3dXBarSpacing	**double**	**80.0**	**CSG**
XtNxrt3dYBarSpacing	**double**	**80.0**	**CSG**

Specifies the amount of space used to display bars (in the X or Y direction) as a percentage of the total amount of space available. Values between 1.0 and 100.0 are valid. A value of 100.0 will cause the bars to abut one another if using regular data (they may not abut if using irregular data). This resource only applies when `XtNxrt3dType` is set to `XRT3D_TYPE_BAR`, and `XtNxrt3d[XY]BarFormat` is `XRT3D_BAR_FIXED`.

XtNxrt3dXDataLabels	**char ****	**NULL**	**CSG**
XtNxrt3dYDataLabels	**char ****	**NULL**	**CSG**

This resource is used to supply the labels that will be used when the `XtNxrt3d[XY]AnnoMethod` is set to `XRT3D_ANNO_DATA_LABELS`. This is a `NULL`-terminated list of strings, where each label is applied to a line of data in the grid.

Note that data labels are not used in scatter plots.

XtNxrt3dXGridLines	**int**	**0**	**CSG**
XtNxrt3dYGridLines	**int**	**0**	**CSG**
XtNxrt3dZGridLines	**int**	**0**	**CSG**

These resources can be used to specify grid lines on each applicable plane. The three planes are represented by constant values, and combined together to specify the affected planes exactly. In 2D, any non-zero value causes grid lines to be drawn perpendicular to the axis. The grid lines are drawn wherever an axis label would be drawn. The following constants are defined in *<Xm/xrt3d_common.h>*, and should be used when setting the value of these resources:

```
#define XRT3D_XY_PLANE 1
#define XRT3D_XZ_PLANE 2
#define XRT3D_YZ_PLANE 4
```

For example, to draw X grid lines on all possible planes, set `XtNxrt3dXGridLines` to:

```
XRT3D_XY_PLANE | XRT3D_XZ_PLANE
```

To remove grid lines, set these resources to zero.

XtNxrt3d[XYZ]GridLineStyle	**Xrt3dLineStyle ***	**solid line**	**CSG**

Controls the color, width, and line pattern in which the X, Y or Z grid lines are drawn. A side effect of setting this resource is to set the corresponding `XtNxrt3d[XYZ]GridLineStyle` resource to `False`.

XtNxrt3d[XYZ]GridLineStyleUseDefault	**Boolean**	**True**	**CSG**

If True, specifies that the default X, Y or Z grid line style is used. The default line style is a solid line of width 1 that is either in the graph foreground color (if specified) or the widget foreground color. Setting the corresponding `XtNxrt3d[XYZ]GridLineStyle` resource has the side effect of setting this resource to `False`.

XtNxrt3dXMax	**double**	***dynamic***	**CSG**
XtNxrt3dYMax	**double**	***dynamic***	**CSG**
XtNxrt3dZMax	**double**	***dynamic***	**CSG**

Specifies the value of the high end of the X, Y, and Z data displayed. X and Y data values higher than this value are not displayed. To zoom into the data, set narrower X and Y minimums and maximums. `XtNxrt3dZMax` cannot be less than the maximum Z data value.[1] A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrt3dXMaxUseDefault	**Boolean**	**True**	**CSG**
XtNxrt3dYMaxUseDefault	**Boolean**	**True**	**CSG**
XtNxrt3dZMaxUseDefault	**Boolean**	**True**	**CSG**

When `True`, the value of `XtNxrt3d[XYZ]Max` is determined by XRT/3d at render time. When an explicit value for `XtNxrt3d[XYZ]Max` is provided, XRT/3d sets this resource to `False`. You cannot set this resource to `False` until a value for `XtNxrt3dZMax` has been calculated or provided. If the value of `XtNxrt3dZMax` is determined to be less than the maximum Z-value of the data, the value of `XtNxrt3dZMaxUseDefault` is forced to `True`.

1. XRT/3d sets the corresponding `UseDefault` resource to `True` when the value of this resource falls inside the Z-range.

XtNxrt3dXMeshFilter	**int**	**1**	**CSG**
XtNxrt3dYMeshFilter	**int**	**1**	**CSG**

Specifies how the mesh is filtered before being displayed. For a value *n*, every *nth* mesh line is drawn. When set to 0, XRT/3d automatically determines a pleasing mesh filter. The filter applies to both regular and irregular grids.

These resources affects the display only when DrawMesh is `True`. They are ignored when the value of `XtNxrt3dType` is `XRT3D_TYPE_BAR` and `XRT3D_TYPE_SCATTER`.

XtNxrt3dXMeshShow	**Boolean**	**True**	**CSG**
XtNxrt3dYMeshShow	**Boolean**	**True**	**CSG**

Specifies whether XRT/3d should draw the mesh lines in the X or Y direction when `XtNxrt3dDrawMesh` is `True`. These resources have no effect on bar charts or scatter plots.

XtNxrt3dXMin	**double**	***dynamic***	**CSG**
XtNxrt3dYMin	**double**	***dynamic***	**CSG**
XtNxrt3dZMin	**double**	***dynamic***	**CSG**

Specifies the value of the low end of the X, Y, and Z data displayed. X and Y data values lower than this value are not displayed. To zoom into the data, set narrower X and Y minimums and maximums. `XtNxrt3dZMin` cannot be greater than the minimum Z data value.[1] A side-effect of setting these resources is that the corresponding `UseDefault` resource is set to `False`.

XtNxrt3dXMinUseDefault	**Boolean**	**True**	**CSG**
XtNxrt3dYMinUseDefault	**Boolean**	**True**	**CSG**
XtNxrt3dZMinUseDefault	**Boolean**	**True**	**CSG**

When `True`, the value of `XtNxrt3d[XYZ]Min` is determined by XRT/3d at render time. When an explicit value for `XtNxrt3d[XYZ]Min` is provided, XRT/3d sets this resource to `False`. You cannot set this resource to `False` until a value for `XtNxrt3dZMin` has been calculated or provided. If the value of `XtNxrt3dZMin` is determined to be greater than the minimum Z-value of the data, the value of `XtNxrt3dZMinUseDefault` is forced to `True`.

XtNxrt3dXRotation	**double**	**45.0**	**CSG**
XtNxrt3dYRotation	**double**	**0.0**	**CSG**
XtNxrt3dZRotation	**double**	**45.0**	**CSG**

Specifies the number of degrees of rotation of the unit cube, in a counter-clockwise direction about the axis. XRT/3d applies rotations in Z-Y-X order. These resources are updated if the user dynamically rotates their view. See section 7.4.2 on page 76 for information on dynamic rotations.

1. XRT/3d sets the corresponding `UseDefault` resource to `True` when the value of this resource falls inside the Z-range.

XtNxrt3dXScale	**double**	**1.0**	**CSG**
XtNxrt3dYScale	**double**	**1.0**	**CSG**
XtNxrt3dZScale	**double**	**1.0**	**CSG**

The `XtNxrt3d[XYZ]Scale` resources can be used to scale the unit cube. For example, to scale the unit cube up by a factor of two in the Y direction (resulting in an oblong unit cube, twice as long in the Y direction as in the X and Z directions), set `XtNxrt3dYScale` to 2.0.

XtNxrt3dXValueLabels	**Xrt3dValueLabel ****	**NULL**	**CSG**
XtNxrt3dYValueLabels	**Xrt3dValueLabel ****	**NULL**	**CSG**
XtNxrt3dZValueLabels	**Xrt3dValueLabel ****	**NULL**	**CSG**

This resource is used to specify labels for the axis to be used when `XtNxrt3d[XY]AnnoMethod` is `XRT3D_ANNO_VALUE_LABELS`. These labels provide a set of strings that are applied at specific values along the axis. See section 3.12 on page 51 for more information.

XtNxrt3dXYColors	**Xrt3dXYColor ****	**NULL**	**CSG**

Specifies a list of colors that are to be applied, in order, to bar charts when `XtNxrt3dDrawShaded` is `True` and `XtNxrt3dDrawZones` is `False`. These colors can be applied to an entire row or column of bar charts, or to an individual cell. See the description of the `Xrt3dXYColor` structure in section 3.3 on page 39 for more information.

This resource has no effect when `XtNxrt3dType` is `XRT3D_TYPE_SURFACE` or `XRT3D_TYPE_SCATTER`.

XtNxrt3dZOrigin	**double**	**0.0**	**CSG**

Defines the Z-value from which bars are filled when `XtNxrt3dType` is `XRT3D_TYPE_BAR` and from which scatter plot drop lines are drawn when `XtNxrt3dType` is `XRT3D_TYPE_SCATTER`. This resource has no effect on the graph when `XtNxrt3dType` is `XRT3D_TYPE_SURFACE`.

XtNxrt3dZoneData	**Xrt3dData ***	**NULL**	**CSG**

When `XtNxrt3dType` is `XRT3D_TYPE_SURFACE`, this resource specifies the zone (i.e. filled contour) data to be displayed draped over surface data (which is specified with `XtNxrt3dSurfaceData`) in a 4D graph. This resource should be set to `NULL` unless projections or a 4D graph is desired. See section 8.1 on page 87 for information on 4D surface graphs.

When `XtNxrt3dType` is `XRT3D_TYPE_BAR`, this resource specifies a dataset which is used to determine bar colors, resulting in a 4D bar chart. See section 8.2 for more information on 4D bar charts.

This resource has no effect when the `XtNxrt3dType` is `XRT3D_TYPE_SCATTER`.

XtNxrt3dZoneMethod	**Xrt3dZoneMethod**	**XRT3D_ZONE_CONTOURS**	**CSG**

For surface and bar charts, specifies the method used to fill the zone regions, displayed when DrawZones is `True`. When set to `XRT3D_ZONE_CONTOURS`, XRT/3d fills between each contour line. When set to `XRT3D_ZONE_CELLS`, XRT/3d fills entire cells in a surface grid, based on the average value in the cell.

A.3 Inherited Resources

XRT/3d inherits resources from its parent classes. The following table alphabetically lists the resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

Resource	Inherited From
XmNdestroyCallback	Core
XmNheight	Core
XmNhighlightColor	XmPrimitive
XmNhighlightOnEnter	XmPrimitive
XmNhighlightPixmap	XmPrimitive
XmNnavigationType	XmPrimitive
XmNshadowThickness	XmPrimitive

Resource	Inherited From
XmNtranslations	Core
XmNtraversalOn	XmPrimitive
XmNuserData	XmPrimitive
XmNwidth	Core
XmNx	Core
XmNy	Core

B

XRT/3d Text Object Resource Reference

Object Synopsis
Text Object Resources
Inherited Resources

This section lists all of the XRT/3d text object resources in alphabetical order. Listed after the resource name is its data type, default value and a list of the Xt procedures (**C**reate, **S**et, **G**et) that may be used with the resource.

B.1 Object Synopsis

Include File:	Not necessary, included with XRT/3d (*<Xm/Xrt3d.h>*)
Class Name:	XtXrt3dText
Class Hierarchy:	Object → XtXrt3dText
Class Pointer:	xtXrt3dTextObjectClass
Instantiation:	`widget = XtCreateWidget(name,xtXrt3dTextObjectClass...)`
Functions/Macros:	*XtCreateXrt3dText()*, *XtIsXrt3dText()*

B.2 Text Object Resources

XtNxrt3dTextAdjust **Xrt3dAdjust** **XRT3D_ADJUST_CENTER** **CSG**

Specifies how multiple lines of text should be aligned within the text object. Valid values are `XRT3D_ADJUST_LEFT`, `XRT3D_ADJUST_CENTER` and `XRT3D_ADJUST_RIGHT`.

XtNxrt3dTextAttachIndexX **int** **0** **CSG**
XtNxrt3dTextAttachIndexY **int** **0** **CSG**

Specifies the grid index that the text object is attached to. This resource affects text object positioning when XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_INDEX. Not used for scatter plots.

XtNxrt3dTextAttachIndexPoint **int** **0** **CSG**

Specifies the point that the text object is attached to. This resource affects text object positioning when the data is of type XRT3D_DATA_POINT.

XtNxrt3dTextAttachIndexSeries **int** **0** **CSG**

Specifies the series that the text object is attached to. This resource affects text object positioning when the data is of type XRT3D_DATA_POINT.

XtNxrt3dTextAttachMethod **Xrt3dTextAttachMethod** **XRT3D_TEXT_ATTACH_INDEX** **CSG**

Specifies how the text object is attached to the graph. Valid values are XRT3D_TEXT_ATTACH_PIXEL, XRT3D_TEXT_ATTACH_POINT and XRT3D_TEXT_ATTACH_INDEX. The positioning of the text object is controlled by resources that correspond to the attachment method: XtNxrt3dTextAttachPixel[XY], XtNxrt3dTextAttachPoint[XYZ], XtNxrt3dTextAttachIndex[XY], XtNxrt3dTextAttachIndexSeries and XtNxrt3dTextAttachIndexPoint.

XtNxrt3dTextAttachPixelX **Position** **0** **CSG**
XtNxrt3dTextAttachPixelY **Position** **0** **CSG**

Specifies the widget pixel coordinates that the text object is attached to. This resource affects text object positioning when XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_PIXEL.

XtNxrt3dTextAttachPointX **double** **0.0** **CSG**
XtNxrt3dTextAttachPointY **double** **0.0** **CSG**
XtNxrt3dTextAttachPointZ **double** **0.0** **CSG**

Specifies the 3D graph coordinates that the text object is attached to. This resource affects text object positioning when XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_POINT.

XtNxrt3dTextBackgroundColor **char *** **NULL** **CSG**

Specifies the text object background color. When XtNxrt3dTextBackgroundColor and XtNxrt3dTextImage 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than black results in white when running on a monochrome server.

XtNxrt3dTextBorder **Xrt3dBorder** **XRT3D_BORDER_NONE** **CSG**

Specifies the style of border used to identify the text object. This resource is only used when the text object is attached by pixel (XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_PIXEL). Valid styles are XRT3D_BORDER_NONE, XRT3D_BORDER_PLAIN, XRT3D_BORDER_3D_IN, XRT3D_BORDER_3D_OUT, XRT3D_BORDER_ETCHED_IN, XRT3D_BORDER_ETCHED_OUT,

XRT3D_BORDER_FRAME_IN, XRT3D_BORDER_FRAME_OUT, XRT3D_BORDER_BEVEL, and XRT3D_BORDER_SHADOW.

Text objects attached by point or index have a 1-pixel line around the object, displayed when the text object background color is not NULL.

XtNxrt3dTextBorderWidth **Dimension** **2** **CSG**

Specifies the width of the text object border when XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_PIXEL. Must be between 0 and 20.

XtNxrt3dTextFont **XmFontList** **"fixed"** **CSG**

Specifies the X font used for the text contained in the text object. This resource is used when XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_PIXEL. XtNxrt3dTextStrokeFont specifies the font used when the text object is attached by point or index.

XtNxrt3dTextForegroundColor **char *** **NULL** **CSG**

Specifies the text object foreground color. 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 2.7 on page 19 for details of color specification.

Setting this color to anything other than white results in black when running on a monochrome server.

XtNxrt3dTextImage **Xrt3dImage** **NULL** **CSG**

Specifies an image to display in the background of the text object. See Background Images on page 38 for more information.

This resource is only used when XtNxrt3dTextAttachMethod is XRT3D_TEXT_ATTACH_PIXEL.

XtNxrt3dTextImageLayout **Xrt3dImageLayout** **XRT3D_IMAGELAYOUT_CENTERED** **CSG**

Specifies the layout of the background image. Valid values are: XRT3D_IMAGELAYOUT_CENTERED and XRT3D_IMAGELAYOUT_TILED.

XtNxrt3dTextImageMinimumSize **Boolean** **False** **CSG**

Specifies that the minimum size of the text area should be determined by the size of the background image.

XtNxrt3dTextLineShow **Boolean** **True** **CSG**

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.

XtNxrt3dTextOffsetX **Position** **0** **CSG**
XtNxrt3dTextOffsetY **Position** **0** **CSG**

Specifies the pixel offset between the center of the text object and the point of attachment. This resource is only used when the text object is either attached to a pixel, or the XRT/3d widget is in a 2D projection.

XtNxrt3dTextPlane **int** **XRT3D_XZ_PLANE** **CSG**

Specifies the orientation of the text object. Text objects must be drawn parallel to one the planes of the cube, and are rendered so that the text is always drawn left-to-right. Valid values are `XRT3D_XZ_PLANE` and `XRT3D_YZ_PLANE`. This resource is ignored in a 2D projection.

XtNxrt3dTextPrintFont **char *** **NULL** **CSG**

Specifies the font used for the text object when printing the XRT/3d widget using *Xrt3dDrawPS()* or *Xrt3dDrawCGM()*. When `NULL`, *Xrt3dDrawPS()* automatically chooses the closest matching PostScript font;[1] *Xrt3dDrawCGM()* uses the Roman Simplex stroke font. This resource can be set to any PostScript font or to a stroke font.[2]

XtNxrt3dTextShow **Boolean** **True** **CSG**

Specifies whether to display the text object.

XtNxrt3dTextStrings **char **** **NULL** **CSG**

Specifies a `NULL`-terminated list of strings displayed in the text object.

XtNxrt3dTextStrokeFont **Xrt3dStrokeFont** **XRT3D_SF_ROMAN_SIMPLEX** **CSG**

Specifies the stroke font used for the text contained in the text object. This resource is used when `XtNxrt3dTextAttachMethod` is `XRT3D_TEXT_ATTACH_POINT` or `XRT3D_TEXT_ATTACH_INDEX`. This resource can be set to any of the valid stroke fonts listed in Appendix E on page page 200. `XtNxrt3dTextFont` specifies the X font used when the text object is attached by pixel.

XtNxrt3dTextStrokeSize **Dimension** **80** **CSG**

Specifies the size of the stroke font. The size is measured in thousandths of the unit cube size, and must be between 0 and 1000. See page page 15 for a description of the unit cube.

B.3 Inherited Resources

The XRT/3d text object inherits resources from its parent class. The following lists the resources inherited along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

Resource	Inherited From
XmNdestroyCallback	Object

1. PostScript fonts are chosen from the 35 PostScript fonts available on most PostScript printers (see Figure 51 on page 167 for the complete list).

2. See page 200 in Appendix E for the list of valid stroke fonts.

C

Procedures and Methods Reference

This appendix lists the XRT/3d procedures and methods in alphabetical order. Procedures marked "(source)" are also supplied as source code. Source routines are located in *$XRTHOME/src/3d/util*. If you want to change a source routine, it should be copied to another directory (and linked ahead of XRT/3d if its name is not changed).

Xrt3dArgValToDouble()

Converts an `XtArgVal` to a double. This procedure is the opposite of *Xrt3dDoubleToArgVal()*.

```
double
Xrt3dArgValToDouble(
    XtArgVal    value
    )
```

Xrt3dComputeZValue()

Procedure which is useful when interactively dragging the value of a grid or point location to a new value.

```
double
Xrt3dComputeZValue(
    Widget      graph,
    int         index1, index2,
    int         pixel_x, pixel_y,
    )
```

graph is the graph handle. For grid data, *index1* and *index2* identify the grid cell that is to be changed. For point data, *index1* identifies the series of the point and *index2* identifies the point number of the point that is to be changed. *pixel_x* and *pixel_y* point to the new location of the screen to which the point has been dragged. If the grid cell or point specified (*index1*, *index2*) is a hole, then *Xrt3dComputeZValue()* returns the hole value for the grid. The *feedback3d* demo program illustrates use of this procedure.

Xrt3dContourStylesFromFile()
Procedure which creates a NULL-terminated array of pointers to Xrt3dContourStyle structures using information read in from a text file. The result can be used for setting the XtNxrt3dContourStyles resource. File names of files that can be read with this routine may be used to set the XtNxrt3dContourStyles resource in resource files.

```
Xrt3dContourStyle **
Xrt3dContourStylesFromFile(
    char    *filename,
    char    *errbuf
    )
```

filename is the pathname of the data file; environment variables are supported. If *Xrt3dContourStylesFromFile()* 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. All other lines define one contour style, and should be of the form:

```
fill-color line-color line-width line-pattern
```

A sample contour style file is in *$XRTHOME/src/3d/samples/100grays.col*.

An example file:

```
! Example contour styles file
! This file defines 3 contour styles
#001e001effdc black 1 solid
#001e08d4ffdc red 2 dotted
#001e118affdc yellow 1 shortdash
```

Xrt3dDataCopy() (source)
Creates a copy of the supplied data set.

```
Xrt3dData *
Xrt3dDataCopy(
    Xrt3dData *source
    )
```

Xrt3dDataShaded() (source)
Creates a new Xrt3dData structure from an existing structure, filling it with a shaded-relief map of the original dataset, suitable for draping over the original data as a 4D dataset.

```
Xrt3dData *
Xrt3dDataShaded(
    Xrt3dData   *source,
    double      sweep,
    double      rise,
    double      scale,
    double      ambient,
    double      intensity
    )
```

The source data can be either a regular or irregular grid. This function calculates reflectance values for each of the facets in the original surface and stores the values in

a new grid. This grid, when viewed with a set of grey-scale contour styles, resembles a shaded-relief map of the original data. The light source is located by the two angles *sweep* and *rise*. The difference in units between the "ground" coordinates in the XY plane and those of the Z values can be controlled through the *scale* parameter. If *scale* is 0, a default value of *scale* will be computed based on the average slope of the facets. The base amount of light for the surface is called the *ambient* light, while the brightness of the light is controlled through the *intensity*; both of these values should be between 0 and 1.

Xrt3dDataSmooth() (source)
Passes a center-weighted averaging function over the grid, resulting in a smoother-looking dataset.

```
void
Xrt3dDataSmooth(
    Xrt3dData   *source,
    double      center_weight
    )
```

The source data can be either a regular or irregular grid. *center_weight* must be between 0 and 1. This procedure is a nine-point filter using all points adjacent to the center. Each new value is a weighted average of the original value and its surrounding points. The weights of the surrounding points are computed based on the value of *center_weight* and its distance from the center point.

Xrt3dDataUpdateDataValue() (source)
Changes a data value to a new value throughout an `Xrt3dData` structure. Can be used on both regular and irregular data grids and on point data. Replaces Z-values only.

```
int
Xrt3dDataUpdateDataValue(
    Xrt3dData   *data,
    double      oldValue,
    double      newValue
    )
```

oldValue is the data value to update; *oldValue* is the new value. *Xrt3dDataUpdateDataValue()* returns 0 on failure and 1 on success.

Xrt3dDataWindow() (source)
Creates a new `Xrt3dData` structure from an existing structure, extracting a subset of the original, and resampling the data to the requested specifications.

```
Xrt3dData *
Xrt3dDataWindow(
    Xrt3dData           *source,
    double              xstart, ystart,
    double              xend, yend,
    int                 numx, numy,
    Xrt3dInterpMethod   interp
    )
```

The source data can be either a regular or irregular grid. The new dataset is a regularly-gridded dataset, with its origin at (*xstart*, *ystart*), *numx* lines in the *x*-

direction, and *numy* lines in the *y*-direction in a region ending at (*xend*, *yend*). Any requested portion that is outside of the region defined by the source grid is filled with holes. *interp* specifies either cubic (XRT3D_INTERP_CUBIC_SPLINE) or linear (XRT3D_INTERP_LINEAR_SPLINE) spline interpolation.

Xrt3dDestroyData()

Destroys an Xrt3dData structure.

```
void
Xrt3dDestroyData(
    Xrt3dData   *data,
    Boolean     all
    )
```

data is the structure to be destroyed. If *all* is True, then any space allocated for data values is also freed.

This data structure is detached from any widget that is currently using it.

Xrt3dDistnIndex()

Calculates the index of the entry in the distribution table for this value. The returned value can be used to access the contour style used for *value* by calling *Xrt3dGetNthContourStyle()*.

```
int
Xrt3dDistnIndex(
    Widget      graph,
    double      value
    )
```

graph is the graph handle.

Xrt3dDistnTableFromFile()

Procedure which creates a pointer to Xrt3dDistnTable structures using information read in from a text file. The result can be used for setting the XtNxrt3dDistnTable resource. File names of files that can be read with this routine may be used to set the XtNxrt3dDistnTable resource in resource files.

```
Xrt3dDistnTable *
Xrt3dDistnTableFromFile(
    char    *filename,
    char    *errbuf
    )
```

filename is the pathname of the data file; environment variables are supported. If *Xrt3dDistnTableFromFile()* 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. All other lines define one distribution table entry.

An example file:

```
! Example distribution table file
! Note the not-quite-linear distribution!
1.0
1.2
1.4
3.0
```

Xrt3dDoubleToArgVal()

Converts a double value to `XtArgVal`, for use when setting any of XRT/3d's floating-point resources.

```
XtArgVal
Xrt3dDoubleToArgVal(
    double  value
    )
```

Xrt3dDrawCGM()

Generates a representation of the graph image in Computer Graphics Metafile (CGM version 1) format. The file is also compatible with CGM versions 2 and 3. CGM files can be sent to a CGM plotter or imported into documents. The CGM file contains the complete graph image, even if it is partially obscured on the screen.

```
int
Xrt3dDrawCGM(
    XtXrt3dWidget       graph,
    FILE                *fp,
    char                *msg,
    double              metric,
    Xrt3dStrokeFont     header_font,
    Xrt3dStrokeFont     footer_font,
    Xrt3dStrokeFont     legend_font
    )
```

graph is the handle of the graph to be output. *fp* should point to a file that is already opened and ready for writing. If *Xrt3dDrawCGM()* encounters problems, it returns a message in *msg* (*msg* should be at least 100 bytes). If *msg* is `NULL` any error message will be written to *stderr*. *Xrt3dDrawCGM()* returns 0 on failure and 1 on success.

metric controls the size of the output–when 0.0, the output device controls the image size; when non-zero, *metric* specifies the number of device units per millimeter. For example, setting *metric* to 0.5 on a 300x600-pixel graph generates a 150x300 mm CGM file.

header_font, *footer_font*, and *legend_font* specify the stroke font replacements for the header, footer, and legend because X fonts cannot be mapped directly to CGM character sets. *Xrt3dDrawCGM()* determines the size of the font using the X font metrics (`Xrt3dStrokeFonts` are defined in Appendix E on page 193).

The following code illustrates the use of *Xrt3dDrawCGM()*:

```
char msg[100];
FILE *fp;
Xrt3dDrawCGM(mygraph, fp, msg,
        0.0,                    /* Don't set scale */
        XRT3D_SF_ITALIC_COMPLEX, /* Header font */
        XRT3D_SF_ROMAN_DUPLEX,   /* Footer font */
        XRT3D_SF_ROMAN_SIMPLEX); /* Legend font */
```

NOTE: This feature is no longer supported as of XRT/3d 3.0.0. The method remains for backward compatibility, but is not guaranteed to work on any features later than XRT/3d 2.1.1.

Xrt3dDrawPS()

Writes an encapsulated PostScript (EPSF-2.0) image of the graph to a file. This procedure is recommended (instead of *Xrt3dOutputPS()*). *Xrt3dDrawPS()* generates a true PostScript file using device-independent PostScript operators, while *Xrt3dOutputPS()* outputs the pixels on the screen using the PostScript *image* operator. PostScript files generated by *Xrt3dDrawPS()* print at the full resolution of the PostScript printer.

Note that the widget must be realized (but not necessarily mapped to the display) before calling this procedure.

```
int
Xrt3dDrawPS(
    Widget  graph,
    FILE    *fp,
    char    *msg,
    int     inches,
    double  paper_width, paper_height, margin,
    int     landscape,
    double  x_offset, y_offset, width, height,
    char    *header_font, *footer_font,
    char    *legend_font,
    int     fill_background,
    int     smart_mono,
    int     showpage
    )
```

Xrt3dDrawPS() returns 0 on failure and 1 on success. *graph* is the handle of the XRT/3d widget to be output. *fp* should point to a file which is already open and ready for writing. If *Xrt3dDrawPS()* 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).

Named PostScript fonts may be specified for the header, footer and legend. If `NULL` is specified, *Xrt3dDrawPS()* 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 *Xrt3dDrawPS()*. For example, *header_font* can be specified as “Times-Roman”.

```
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 51 The 35 Standard LaserWriter II Font Names

If *fill_background* is `False`, *Xrt3dDrawPS()* will not draw the background color.

If *smart_mono* is XRT3D_PS_COLOR_AS_IS, colors print in the color used on the screen; if XRT3D_PS_COLOR_MONO, colors are mapped to distinctive gray shades; if XRT3D_PS_COLOR_AUTO, the code detects[1] whether the output device is color or monochrome and prints colors or gray shades accordingly. This argument is ignored on a monochrome X server.

1. Some monochrome PostScript printers (such as the SPARCprinter) incorrectly identify themselves as color printers. To force *smart_mono* output on these printers, use `XRT3D_PS_COLOR_MONO`.

If *showpage* is `True`, *Xrt3dDrawPS()* will put a PostScript *showpage* operator at the end of the EPSF file. It is possible to print several graph images on one page by making several calls to *Xrt3dDrawPS()* using the same file pointer, and setting showpage to `False` in all but the last call.

The following call will print a graph image as large as possible while still maintaining the aspect ratio, on a standard letter-size page with a half-inch margin. All fonts will default.

```
char msg[100];
FILE *fp;
Xrt3dDrawPS(mygraph, fp, msg,
    True,                   /* Use inches */
    8.5, 11.0, 0.5,         /* 8.5x11, 1/4" margin*/
    False,                  /* Portrait */
    0.0, 0.0, 0.0, 0.0,     /* Full Size */,
    NULL, NULL, NULL,       /* Default Fonts */
    True,                   /* Keep background */
    XRT3D_PS_COLOR_AS_IS,   /* Don't adjust */
    True);                  /* Show the page */
```

Xrt3dDupContourStyles()

Duplicates an array of `Xrt3dContourStyle` structures. The pointer to the duplicate array is returned.

```
Xrt3dContourStyle **
Xrt3dDupContourStyles(
    Xrt3dContourStyle **cs
    )
```

Xrt3dDupDataStyles()

Duplicates an array of `Xrt3dDataStyle` structures.

```
Xrt3dDataStyle **
Xrt3dDupDataStyles(
    Xrt3dDataStyle **ds
    )
```

Xrt3dDupDistnTable()

Duplicates a `Xrt3dDistnTable` structure. The pointer to the duplicate table is returned. `NULL` is returned if the allocation failed for any reason.

```
Xrt3dDistnTable *
Xrt3dDupDistnTable(
    Xrt3dDistnTable *dt
    )
```

Xrt3dDupStrings()

Duplicates an array of strings. The pointer to the duplicate array is returned. `NULL` is returned if the allocation failed for any reason.

```
char **
Xrt3dDupStrings(
    char    **s
    )
```

Xrt3dDupValueLabels()

Duplicates a list of value labels.

```
Xrt3dValueLabel **
Xrt3dDupValueLabels(
    Xrt3dValueLabel **vlabels
    )
```

vlabels is an array of NULL-terminated pointers to structures of type Xrt3dValueLabel. A copy of this list is returned to the caller. NULL is returned if the allocation failed for any reason.

Xrt3dDupXYColors()

Duplicates a list of XYColors.

```
Xrt3dXYColor **
Xrt3dDupXYColors(
    Xrt3dXYColor **xycols
    )
```

xycols is an array of NULL-terminated pointers to structures of type Xrt3dXYColor. A copy of this list is returned to the caller. NULL is returned if the allocation failed for any reason.

Xrt3dEnableImageType()

Registers a new image type with XRT/3d. 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. xtXrt3dWidgetClass). *functions* is a pointer to an XrtPDSImageFuncs structure (see Registering Image Types on page 106 for details).

Xrt3dFreeContourStyles()

Frees the memory used by an array of Xrt3dContourStyle structures.

```
void
Xrt3dFreeContourStyles(
    Xrt3dContourStyle **cs
    )
```

Xrt3dFreeDataStyles()

Frees the memory used by an array of Xrt3dDataStyle structures.

```
void
Xrt3dFreeDataStyles(
    Xrt3dDataStyle **ds
    )
```

Xrt3dFreeDistnTable()

Frees the memory used by an `Xrt3dDistnTable` structure.

```
void
Xrt3dFreeDistnTable(
    Xrt3dDistnTable *dt
    )
```

Xrt3dFreePropString()

Frees the memory used by a string allocated by *Xrt3dGetPropString()*.

```
void
Xrt3dFreePropString(
    char    *str
    )
```

Xrt3dFreeStrings()

Frees the memory used by an array of strings.

```
void
Xrt3dFreeStrings(
    char    **s
    )
```

Xrt3dFreeValueLabels()

Frees a list of value labels.

```
void
Xrt3dFreeValueLabels(
    Xrt3dValueLabel **vlabels
    )
```

vlabels is an array of `NULL`-terminated pointers to structures of type `Xrt3dValueLabel`. The memory occupied by these structures, and the pointer array, is freed.

Never free an `Xrt3dValueLabel` list that is currently being used by the widget.

Xrt3dFreeXYColors()

Frees a list of XYColors.

```
void
Xrt3dFreeXYColors(xycols)
    Xrt3dXYColors **xycols;
```

xycols is an array of `NULL`-terminated pointers to structures of type `Xrt3dXYColor`. The memory occupied by these structures, and the pointer array, is freed.

Never free an `Xrt3dXYColor` list that is currently being used by the widget.

Xrt3dGetDistnTable()

Returns a pointer to the distribution table currently being used by the widget.

```
Xrt3dDistnTable *
Xrt3dGetDistnTable(
    Widget  graph
    )
```

This procedure allocates space for the returned table, which may be freed by the application by calling *Xrt3dFreeDistnTable()*. Returns `NULL` on failure for any reason.

Xrt3dGetImage()

Returns a pointer to the XImage of a XRT/3d widget. This may be useful for Xlib-level programmers who need access to the widget's XImage.

```
XImage *
Xrt3dGetImage(
    Widget  graph
    )
```

graph is the handle of the XRT/3d widget to be output. *Xrt3dGetImage()* will return `NULL` on failure. The XImage will be complete, even if the widget is partially obscured on the screen.

Xrt3dGetNthContourStyle()

Returns a pointer to a contour style.

```
Xrt3dContourStyle *
Xrt3dGetNthContourStyle(
    Widget      graph,
    int         index,
    Boolean     used
    )
```

graph is the graph handle. The (zero-based) *index* specifies which contour style should be returned. Remember that the subset of contour styles actually used will depend on the number of levels (*nlevels*) in the distribution. Set *used* to `True` to lookup contour styles from the set of contour styles actually being used. Set *used* to `False` to lookup contour styles from the set of all contour styles currently defined. The pointer returned by this method should be considered read-only.

Xrt3dGetNthDataLabel()

Returns a pointer to a data label.

```
char *
Xrt3dGetNthDataLabel(
    Widget      graph,
    Xrt3dAxis   axis,
    int         index
    )
```

graph is the graph handle. *axis* identifies the axis the label is to be taken from. *index* is the 0-indexed number of the requested label. If there are no data labels for this axis, or if *index* is out of range, *Xrt3dGetNthDataLabel()* returns `NULL`.

Xrt3dGetNthDataStyle()

Returns a pointer to the *n*th data style currently being used by the graph.

```
Xrt3dGetNthDataStyle(
    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*.

Xrt3dGetNthFooterString()

Returns a pointer to the *n*th footer string currently being used by the graph.

```
char *
Xrt3dGetNthFooterString(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which footer string should be returned. The default footer string for *index* will be returned if *index* is larger than the number of currently defined footer strings. The pointer returned by this method should be considered read-only.

Xrt3dGetNthHeaderString()

Returns a pointer to the *n*th header string currently being used by the graph.

```
char *
Xrt3dGetNthHeaderString(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which header string should be returned. The default header string for *index* will be returned if *index* is larger than the number of currently defined header strings. The pointer returned by this method should be considered read-only.

Xrt3dGetNthLegendString()

Returns a pointer to the *n*th legend string currently being used by the graph.

```
char *
Xrt3dGetNthLegendString(
    Widget      graph,
    int         index
    )
```

graph is the graph handle. The (zero-based) *index* specifies which legend string should be returned. The default legend string for *index* will be returned if *index* is larger than the number of currently defined legend strings. The pointer returned by this method should be considered read-only.

Xrt3dGetPropString()

Retrieves the current value of a graph resource as a string.

```
Boolean
Xrt3dGetPropString(
    Widget      graph,
    int         resource,
    char        **str          /* Returned */
    )
```

graph is the graph handle. *resource* specifies any XRT/3d resource (using its cross-platform name listed on page page 205); *str* is a pointer to a string returned with the string representation of the resource value. *str* must be freed after use by calling *Xrt3dFreePropString()*.

Xrt3dGetValueLabel()

Returns a pointer to the value label which has the closest value to the supplied value label.

```
Xrt3dValueLabel *
Xrt3dGetValueLabel(
    Widget            graph,
    Xrt3dAxis         axis,
    Xrt3dValueLabel   *vlabel
    )
```

graph is the graph handle. *axis* identifies the axis the label is to be taken from. *vlabel* points to an `Xrt3dValueLabel` structure which has the *value* element filled in. A pointer to the value label which lies closest to the requested value is returned. If there are no value labels for this *axis*, *Xrt3dGetValueLabel()* returns `NULL`.

Xrt3dGetValues()

Retrieves the current value of one or more graph resources.

```
void
Xrt3dGetValues(
    Widget      graph,
    ...,
    NULL
    )
```

graph is the graph handle. ... is one or more *resource-value* pairs. *resource* is any XRT/3d resource (specified as its `Xrt3dProperty` cross-platform name listed on page page 205); *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 *Xrt3dArgValToDouble()* to access resources of type `double`. The list is terminated by `NULL`.

Xrt3dGetXYColor()

Returns a pointer to a color in a bar chart.

```
char *
Xrt3dGetXYColor(
    Widget  graph,
    int     xindex, yindex
    )
```

graph is the graph handle. *xindex* and *yindex* specify the bar or set of bars to retrieve the color from.

Xrt3dInsertNthDataStyle()

Inserts a new data style which becomes the *n*th data style.

```
int
Xrt3dInsertNthDataStyle(
    Widget          graph,
    int             index
    Xrt3dDataStyle  *ds
    )
```

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 `Xrt3dDataStyle` structure containing the new data style to be used. XRT/3d makes its own copy of the data style pointed to by *ds*.

A side effect of this method is that the appropriate `XtNxrt3dDataStylesUseDefault` resource is set to False.

Xrt3dMakeDataFromFile()

Procedure which will allocate an `Xrt3dData` structure and load it with data read in from a text file. See also *Xrt3dSaveDataToFile()*.

```
Xrt3dData *
Xrt3dMakeDataFromFile(
    char    *filename,
    char    *errbuf
    )
```

filename is the pathname of the data file; this may include an environment variable. If *Xrt3dMakeDataFromFile()* 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 the data type, which may be `GRID`, `IRGRID` or `POINT`. (For `GRID` and `IRGRID`, this is followed by two integers (which should be ≥ 1) specifying the number of points in the X and Y directions. For POINT, this is followed by one integer (which should be > 1) that specifies the number of series.) The next value is a floating-point number specifying the data hole value.

If `GRID` was specified, the next two floating-point numbers specify the grid step amount in the X and Y direction. The grid step amounts are followed by two floating-point numbers which indicate the X and Y grid origin values. Finally, the surface data values are listed in order of increasing Y values. (All of the Y values for X=origin are listed. Then all the Y values for X = (x origin + x grid step) are listed, etc.)

An example of a suitable `GRID` data file:

```
! Grid has 50 by 30 points
! Holes have value 100.0
! Grid increases in
!       X steps of 1.0 & Y steps of 2.0
! Origin of grid is at X = -20.0, Y = 50.0
GRID 50 30
100.0 1.0 2.0 -20.0 50.0
! 1500 data values follow, one for each grid pt.
49.875000 43.765625 38.500000 33.984375 30.12400
26.828125 24.000000 21.546875 19.375000 17.39062
. . .
```

If `IRGRID` is specified, then a list of xgrid and ygrid values is supplied instead of the origin, xstep and ystep values.

An example of a suitable `IRGRID` data file:

```
! Irregular grid has 10 by 5 points
! Holes have value 100.0
! Ten x values are given
! Five y values are given
IRGRID 10 5
100.0
20 21.1 22.3 23 24.4 25.4 27.8 29.9 30.1 31.2
50.3 51.3 52.6 54.8 59.6
! 50 data values follow
23.34343 12.89239 11.99423 15.781212 18.81988
. . .
```

If `POINT` is specified, a list of series data is then supplied. Each series is specified by an integer which gives the number of points in the series and then a list of (x, y, z) values. For example:

```
!Point data has 3 series.
!Holes have value -1000
POINT 3
-1000
!Series 1 has 5 points. 15 (x, y, z) values follow.
5
0.75 0.50 0.30
0.45 0.50 0.30
0.75 0.40 0.30
0.75 0.50 0.40
0.45 0.40 0.40
!Series 2 has 3 points.  9 (x, y, z) values follow.
3
1.2 1.3 1.4
1.5 1.6 1.7
1.8 1.9 2.0
!Series 3 has 4 points. 12 (x, y, z) values follow.
4
-3.0 -2.0 -1.0
 0.0  1.0  2.0
 2.0  1.0  0.0
-1.0 -2.0 -3.0
```

Xrt3dMakeGridData()

Procedure used to allocate an empty `Xrt3dData` structure for regularly gridded data. To allocate space for irregularly gridded data, use *Xrt3dMakeIrGridData()*.

```
Xrt3dData *
Xrt3dMakeGridData(
    int     numx, numy,
    double  noval,
    double  xstep, ystep,
    double  xorig, yorig,
    Boolean all
    )
```

numx and *numy* specify the number of elements in the grid in the X and Y directions and should always be ≥ 1. *noval* defines the value that XRT/3d will treat as a data hole. *xstep* and *ystep* define the step between grid elements along the X and Y axis. *xorig* and *yorig* define the origin of the grid in axis coordinates. If *all* is `True`, then space for the data values is also allocated. Otherwise, only the structure itself is allocated, and the passed values are assigned to the structure.

Xrt3dMakeIrGridData()

Procedure used to allocate an empty `Xrt3dData` structure for irregularly gridded data. To allocate space for regularly gridded data, use *Xrt3dMakeGridData()*.

```
Xrt3dData *
Xrt3dMakeIrGridData(
    int         numx, numy,
    double      noval,
    Boolean     all
    )
```

numx and *numy* specify the number of elements in the grid in the X and Y directions and should always be ≥ 1. *noval* defines the value that XRT/3d will treat as a data hole. If *all* is `True`, then space for the data values, and the *xgrid* and *ygrid* arrays is also allocated. Otherwise, only the structure itself is allocated, and the passed values are assigned to the structure.

Xrt3dMakePointData()

Procedure used to allocate an empty `Xrt3dData` structure for point data. Memory for an `Xrt3dData` structure is allocated along with an array of *nseries* `Xrt3dPointSeries` structures.

```
Xrt3dData*
Xrt3dMakePointData(
    int         nseries,
    int         npoints,
    double      noval,
    Boolean     all
    )
```

nseries specifies the number of `Xrt3dData` structures to allocate and should always be ≥ 1. If *all* is `True`, then for each series, *npoints* `Xrt3dPoint3D` structures are allocated where `Xrt3dPoint3D` is defined as follows:

```
typedef struct {
     double    x, y, z;
} Xrt3dPoint3D;
```

Xrt3dMakePointDataSeries()

Allocates an array of *npoints* `Xrt3dPoint3D` structures to be stored in the passed in `Xrt3dPointSeries` structure, freeing any previous points beforehand. The `Xrt3dPoint3D` structure is defined as follows:

```
typedef struct {
     double    x, y, z;
} Xrt3dPoint3D;

Boolean
Xrt3dMakePointDataSeries(
    Xrt3dPointSeries    *pseries,
    int                 npoints,
    )
```

pseries is a pointer to an `Xrt3dPointSeries` structure; *npoints* is the number of points to allocate.

Xrt3dMap()

Maps a pixel coordinate to a graph coordinate.

```
Xrt3dRegion
Xrt3dMap(
    Widget          graph,
    Position        pix_x, pix_y,
    Xrt3dMapResult  *map
    )
```

Used by an application to determine the graph coordinates corresponding to *pix_x* and *pix_y* (for the graph whose handle is specified by *graph*). Results are returned in *map*. Typically used in an event handling procedure.

Xrt3dMrmInitialize()

Registers the XRT/3d widget class with Mrm. For use by UIL applications. This routine must be called after *MrmInitialize()*.

```
void
Xrt3dMrmInitialize(void)
```

Xrt3dOutputPS()

Outputs an encapsulated PostScript (EPSF) representation of the graph image 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.

Since this procedure creates a “bitmap PostScript” file which will not print at the full resolution of the printing device, it is usually more appropriate to use *Xrt3dDrawPS()* instead of *Xrt3dOutputPS()*.

```
int
Xrt3dOutputPS(
    Widget      graph,
    FILE        *fp,
    Boolean     color,
    char        *msg
    )
```

graph is the handle of the XRT/3d widget to be output. *fp* should point to a file which is already opened and ready for writing. If color is set to `True`, a PostScript *colorimage* will be created, otherwise a monochrome PostScript *image* will be created.

If *Xrt3dOutputPS()* 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*. *Xrt3dOutputPS()* returns 0 on failure and 1 on success. The PostScript file will be complete, even if the widget 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.

Xrt3dOutputXwd()

Outputs an X Window Dump representation of the graph image to a file.

```
int
Xrt3dOutputXwd(
    Widget  graph,
    FILE    *fp,
    char    *msg
    )
```

graph is the handle of the XRT/3d widget to be output. *fp* should point to a file which is already opened and ready for writing. If *Xrt3dOutputXwd()* 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*. *Xrt3dOutputXwd()* returns 0 on failure and 1 on success. The xwd file will be complete, even if the widget is partially obscured on screen.

Xrt3dPick()

Picks the displayed data that is closest to the given pixel coordinate.

```
Xrt3dRegion
Xrt3dPick(
    Widget              graph,
    Position            pix_x, pix_y,
    Xrt3dPickResult     *pick
    )
```

Used by an application to determine what data (in terms of grid point indices) is displayed closest to the pixel at (*pix_x*, *pix_y*) for the graph whose handle is specified by *graph*. Results are returned in *pick*. This is typically used in an event handling procedure.

Xrt3dRemoveNthDataStyle()

Removes the *n*th data style.

```
int
Xrt3dRemoveNthDataStyle(
    Widget       graph,
    int          index,
    )
```

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. *Xrt3dRemoveNthDataStyle*() returns 1 on success, 0 on failure.

Xrt3dResetContourStyles()

Specifies that XRT/3d is to return to using the default contour styles.

```
void
Xrt3dResetContourStyles(
    Widget      graph,
    )
```

graph is the graph handle.

Xrt3dSampleColorFunc()

This function provides an example of a function that can be used for the `XtNxrt3dColorFunc` resource. It is an adaptation of the XPM closest color matching algorithm (adapted to fit the semantics that `XtNxrt3dColorFunc` needs). The source code for this function is provided in *$XRTHOME/src/3d/util/x_color_func.c*.

```
unsigned long
Xrt3dSampleColorFunction(
    Display     *display,
    int         screen,
    Colormap    colormap,
    XColor      *color,
    Boolean     do_free
    )
```

Xrt3dSaveDataToFile()

Procedure which writes out the data stored in an `Xrt3dData` structure to a text file, in a format suitable for use with *Xrt3dMakeDataFromFile()*.

```
int
Xrt3dSaveDataToFile(
    Xrt3dData   *data,
    char        *filename,
    char        *errbuf
    )
```

data must point to a valid `Xrt3dData` structure. *filename* is the pathname of the data file; this may contain an environment variable. If *Xrt3dMakeDataFromFile()* 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. This procedure returns 1 on success and 0 on failure.

Xrt3dSaveImageAsJpeg()

Outputs a JPEG representation of the graph to a file. The JPEG image type must first be registered with XRT/3d using *Xrt3dEnableImageType()*.

```
Boolean
Xrt3dSaveImageAsJpeg(
    Widget          graph,
    String          filename,
    String          msg
    )
```

graph is the graph handle of the graph to be output. *filename* is the name of the file to which the image is to be written. If *Xrt3dSaveImageAsJpeg()* 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. *Xrt3dSaveImageAsJpeg()* returns `True` on success.

Xrt3dSaveImageAsPng()

Outputs a PNG representation of the graph to a file. The PNG image type must first be registered with XRT/3d using *Xrt3dEnableImageType()*.

```
Boolean
Xrt3dSaveImageAsPng(
    Widget          graph,
    String          filename,
    String          msg
    )
```

graph is the graph handle of the graph to be output. *filename* is the name of the file to which the image is to be written. If *Xrt3dSaveImageAsPng()* 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. *Xrt3dSaveImageAsPng()* returns `True` on success.

Xrt3dSaveImageGeneric()

Outputs a representation of the graph to a file based on the image type. The image type must be registered with XRT/3d using *Xrt3dEnableImageType()* unless it is a preregistered type (XPM or XWD).

```
Boolean
Xrt3dSaveImageGeneric(
    Widget          graph,
    int             image_type,
    String          filename,
    String          msg
    )
```

graph is the graph 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 *Xrt3dSaveImageGeneric()* 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. *Xrt3dSaveImageGeneric()* returns `True` on success.

Xrt3dSetNthContourStyle()

Sets the n^{th} contour style.

```
void
Xrt3dSetNthContourStyle(
    Widget              graph,
    int                 index,
    Xrt3dContourStyle   *cs,
    Boolean             used
    )
```

graph is the graph handle. The (zero-based) *index* specifies which contour style should be changed. *cs* points to an `Xrt3dContourStyle` structure containing the new contour style.

Remember that the subset of contour styles actually used will depend on the number of levels (*nlevels*) in the distribution. If *used* is `True`, *index* refers only to the contour

styles in use at this time. If *used* is `False` *index* refers to the array of all contour styles defined.

XRT/3d makes its own copy of the contour style pointed to by *cs*.

Xrt3dSetNthDataLabel()

Sets the n^{th} data label.

```
void
Xrt3dSetNthDataLabel(
    Widget      graph,
    Xrt3dAxis   axis,
    int         index,
    char        *label
    )
```

graph is the graph handle. *axis* identifies the axis to be labelled. *index* is the 0-indexed number of the new label. *label* is the value of the new label. If *index* is greater than the number of labels in use, all intervening labels are assigned an empty string.

Xrt3dSetNthDataStyle()

Sets the n^{th} data style.

```
void
Xrt3dSetNthDataStyle(
    Widget          graph,
    int             index,
    Xrt3dDataStyle  *ds
    )
```

graph is the graph handle. The (zero-based) *index* specifies which data style should be changed. *ds* points to an `Xrt3dDataStyle` structure containing the new data style to be used. XRT/3d makes its own copy of the data style pointed to by ds.

A side effect of this method is that the appropriate `XtNxrt3dDataStylesUseDefault` resource is set to False.

Xrt3dSetNthFooterString()

Sets the n^{th} footer string.

```
void
Xrt3dSetNthFooterString(
    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/3d makes its own copy of the characters pointed to by *s*.

Xrt3dSetNthHeaderString()

Sets the n^{th} header string.

```
void
Xrt3dSetNthHeaderString(
    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/3d makes its own copy of the characters pointed to by *s*.

Xrt3dSetNthLegendString()

Sets the n^{th} legend string.

```
void
Xrt3dSetNthLegendString(
    Widget      graph,
    int         index,
    char        *s
    )
```

graph is the graph handle. The (zero-based) *index* specifies which legend string should be changed. *s* is the new string to be used. XRT/3d makes its own copy of the characters pointed to by *s*.

Xrt3dSetPropString()

Sets a graph resource to the value represented by a string.

```
Boolean
Xrt3dSetPropString(
    Widget          graph,
    Xrt3dProperty   resource,
    char            *str
    )
```

graph is the graph handle. *resource* specifies any XRT/3d resource (using its cross-platform name listed on page page 205); *str* is a pointer to a string representation to set the resource to (syntax is the same as for a resource file, see Appendix G on page 217).

Xrt3dSetReverseConverters()

Registers the reverse converters for use with *XtConvertAndStore()*. Reverse converters allow applications to get resource values in a human-readable form.

```
void
Xrt3dSetReverseConverters(void)
```

Xrt3dSetValueLabel()

Defines, replaces or deletes a value label.

```
void
Xrt3dSetValueLabel(
    Widget            graph,
    Xrt3dAxis         axis,
    Xrt3dValueLabel   *vlabel
    )
```

graph is the graph handle. *axis* identifies the axis to be labelled. *vlabel* points to an `Xrt3dValueLabel` structure which has the *value* and *label* elements filled in. If *value* is the same as an existing value label for this axis, the new label replaces the existing label; if the new label is `NULL`, then the existing value label is deleted.

Due to round-off error locate the value label to be deleted or modified using *Xrt3dGetValueLabel()*.

Xrt3dSetValues()

Sets one or more graph resources.

```
void
Xrt3dSetValues(
    Widget      graph,
    ...,
    NULL
    )
```

graph is the graph handle. ... is one or more *resource-value* pairs. *resource* is any XRT/3d resource (specified as its `Xrt3dProperty` cross-platform name listed on page 205); *value* specifies the new value. The data type of *value* must be the same as the resource's data type. Do not use *Xrt3dDoubleToArgVal()* to process resources of type `double`. The list is terminated by `NULL`.

Xrt3dSetXYColor()

Defines, replaces or deletes a color in a bar chart.

```
void
Xrt3dSetXYColor(
    Widget        graph,
    int           xindex, yindex,
    char          *color
    )
```

graph is the graph handle. *xindex* and *yindex* specify the bar or set of bars to which the *color* will be applied. If *xindex* is -1, *color* is applied to all bars which are along the column defined by *yindex*; if *yindex* is -1, *color* is applied to all bars which are along the row defined by *xindex*; if both *xindex* and *yindex* are -1, the *color* is applied to all the bars in the chart.

Color requests are applied in the order they are specified. For instance, a request to assign the color blue to (-1, 3), followed by a request to assign the color red to (2, 3) results in bars in column 3 being blue, except the bar at *xindex* 2, which is red.

XRT/3d will not keep more than one entry per (*xindex*, *yindex*) in its internal list of XYColors. If *Xrt3dSetXYColor()* is called with a color of `NULL`, the matching entry is

removed from the list. Otherwise, any call to *Xrt3dSetXYColor()* places the new entry at the end of the list, and removes any duplicates from the list.

This procedure only has effect when `XtNxrt3dType` is `XRT3D_TYPE_BAR`, and `XtNxrt3dDrawShaded` is `True`, and `XtNxrt3dDrawZones` is `False`.

Xrt3dTextGetPropString()

Retrieves the current value of a graph resource as a string.

```
Boolean
Xrt3dTextGetPropString(
    Widget      hText,
    int         resource,
    char        **str          /* Returned */
    )
```

hText is the text object handle. *resource* specifies any XRT/3d resource (using its cross-platform name listed on page page 205); *str* is a pointer to a string returned with the string representation of the resource value. *str* must be freed after use by calling *Xrt3dFreePropString()*.

Xrt3dTextGetValues()

Retrieves the current value of one or more text object resources.

```
void
Xrt3dTextGetValues(
    Widget      hText,
    ...,
    NULL
    )
```

hText is a text object handle. ... is one or more *resource-value* pairs. *resource* is any text object resource (specified as its `Xrt3dProperty` cross-platform name listed on page 205); *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. The list is terminated by `NULL`.

Xrt3dTextSetPropString()

Sets a graph resource to the value represented by a string.

```
Boolean
Xrt3dTextSetPropString(
    Widget          hText,
    Xrt3dProperty   resource,
    char            *str
    )
```

hText is the text object handle. *resource* specifies any XRT/3d resource (using its cross-platform name listed on page 205); *str* is a pointer to a string representation to set the resource to (syntax is the same as for a resource file, see Appendix G on page 217).

Xrt3dTextSetValues()

Sets one or more text object resources.

```
void
Xrt3dTextSetValues(
    Widget      hText,
    ...,
    NULL
    )
```

hText is a text object handle. ... is one or more *resource-value* pairs. *resource* is any text object resource (specified as its `Xrt3dProperty` cross-platform name listed on page 205); *value* specifies the new value. The data type of *value* must be the same as the resource's data type. Do not use *Xrt3dDoubleToArgVal()* to access resources of type `double`. The list is terminated by `NULL`.

Xrt3dUnmap()

Unmaps from a graph coordinate to a pixel coordinate.

```
void
Xrt3dUnmap(
    Widget          graph,
    double          x, y, z,
    Xrt3dMapResult  *map
    )
```

Used by an application to determine the pixel coordinates corresponding to the graph coordinate (*x*, *y*, *z*) in the graph whose handle is specified by *graph*. 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.

Xrt3dUnpick()

Determines a pixel coordinate given a point on the surface grid for grid data or a scatter plot point for point data.

```
void
Xrt3dUnpick(
    Widget          graph,
    int             xindex, yindex,
    Xrt3dPickResult *pick
    )
```

Used by an application to determine the pixel coordinates at which a data value is displayed (in the graph whose handle is specified by *graph*). For grid data, *xindex* and *yindex* are the indices of the grid coordinate to be unpicked. For point data, *xindex* refers to the series and *yindex* to the point number of the point to be unpicked.

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.

Xrt3dZInterpolate()

Determines a Z-value of any point on the surface. Does not apply to point data.

```
double
Xrt3dZInterpolate(
    Widget      graph,
    double      x, y
    )
```

This procedure returns an estimate of the surface value at (*x*, *y*) calculated using bilinear interpolation. *graph* is the graph handle. If `XRT3D_SURFACE_DATA` points to a data set of type `XRT3D_DATA_POINT`, returns the hole value.

XtCreateXrt3d()

Creates an XRT/3d widget.

```
Widget
XtCreateXrt3d(
    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()*).

XtCreateXrt3dText()

Creates an XRT/3d text object.

```
Widget
XtCreateXrt3dText(
    Widget  parent,
    char    *name,
    Arg     *arglist,
    int     argcount
    )
```

parent is the parent widget (it must be of class `xtXrt3dWidgetClass`); *name* is the created widget's name; *arglist* is a list of *argcount* arguments (created by *XtSetArg()*).

XtIsXrt3d()

Returns `True` if a widget is a subclass of an XRT/3d widget.

```
Boolean
XtIsXrt3d(
    Widget  graph
    )
```

XtIsXrt3dText()

Returns `True` if a widget is a subclass of an XRT/3d text object.

```
Boolean
XtIsXrt3dText(
    Widget  graph
    )
```

D

Translations and Actions Reference

Default Translations
Action Routines

D.1 Default Translations

XRT/3d inherits translations from XmPrimitive, and defines the following translations of its own:

Event	Action
Shift <Btn2Down>	ModifyStart(Translate) DrawPreview()
Ctrl <Btn2Down>	ModifyStart(Scale) DrawPreview()
~Shift ~Ctrl <Btn2Down>	ModifyStart(Rotate) DrawPreview()
Ctrl <Btn1Down>	ModifyStart(Zoom)
<Key> r	ModifyStart(Reset) ViewReset() ModifyEnd(Reset)
<Key> x	SelectAxis(X) DrawPreview()
<Key> y	SelectAxis(Y) DrawPreview()
<Key> z	SelectAxis(Z) DrawPreview()
<Key> e	SelectAxis(Eye) DrawPreview()
<Key> n	SelectAxis(None) DrawPreview()
Shift <Btn2Motion>	TranslateUpdate() DrawPreview()
Ctrl <Btn2Motion>	ScaleUpdate() DrawPreview()
~Shift ~Ctrl <Btn2Motion>	RotateUpdate() DrawPreview()
Ctrl <Btn1Motion>	ZoomUpdate()

Event	Action
Shift <Btn2Up>	ModifyEnd(Translate)
Ctrl <Btn2Up>	ModifyEnd(Scale)
~Shift ~Ctrl <Btn2Up>	ModifyEnd(Rotate)
Ctrl <Btn1Up>	ZoomEnd() ModifyEnd(Zoom)
Button1 <Key> osfCancel[a]	ZoomCancel() ModifyEnd(Zoom)
Button2 <Key> osfCancel[a]	ModifyCancel() ModifyEnd()
Ctrl <KeyUp>	ZoomCancel()

a. When the XRT/3d 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 actions.

D.2 Action Routines

XRT/3d defines the following action routines:

DrawGraph()
Draws the entire unit cube to provide visual feedback as a user interacts with the widget. By default, this action routine is not bound to any event. It may be used as a replacement for the *DrawPreview()* action.

DrawPreview()
Draws a quick outline of the unit cube to provide visual feedback as a user interacts with the widget. This is the default action for the interactive transformation and rotation actions, providing real-time visual feedback of the effects of the user's actions.

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 *Xrt3dMap()* procedure and triggers the callbacks specified by `XtNxrt3dMapCallback`, passing each callback a pointer to an `Xrt3dMapCallbackStruct` structure that contains the graph coordinates.

ModifyCancel()
Cancels a modification (scaling, translation or rotation) in progress, restoring the resource values to their values before the *ModifyStart()* action was called.

ModifyStart()
Starts any user interaction, giving the application a way to disallow it. *ModifyStart()* calls the callbacks specified by `XtNxrt3dModifyCallback`, passing each callback a pointer to an `Xrt3dModifyCallbackStruct` structure with *reason* set to `XRT3D_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 widget
`Scale`	User wants to scale the widget
`Rotate`	User wants to rotate the widget
`Zoom`	User wants to zoom the widget
`Reset`	User wants to reset the view

ModifyEnd()

Repaints the XRT/3d widget (triggering an `XtNxrt3dxrt3dExposeCallback`). *ModifyEnd()* calls the callbacks specified by `XtNxrt3dModifyCallback`, passing each callback a pointer to an `Xrt3dModifyCallbackStruct` structure with *reason* set to `XRT3D_REASON_MODIFY_END`. *ModifyEnd()* takes an optional parameter that indicates the user interaction completed:

`Translate`	User has finished translating the widget
`Scale`	User has finished scaling the widget
`Rotate`	User has finished rotating the widget
`Zoom`	User has finished zooming the widget
`Reset`	User has finished resetting the view

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 *Xrt3dPick()* procedure and triggers the callbacks specified by `XtNxrt3dPickCallback`, passing each callback a pointer to an `Xrt3dPickCallbackStruct` structure that contains the graph data values.

RotateUpdate()

Rotates 3D graphs. *RotateUpdate()* calculates new values for the `XtNxrt3d[XYZ]Rotation` resources, and calls any callbacks specified by `XtNxrt3dRotateCallback`, passing each callback a pointer to an `Xrt3dRotateCallbackStruct` structure. Unless the application sets

Xrt3dRotateCallbackStruct's *doit* parameter to False, *RotateUpdate()* applies the new graph rotation values to the widget.

ScaleUpdate()

Scales the graph. *ScaleUpdate()* calculates new values for the XtNxrt3dViewScale resource, and calls any callbacks specified by XtNxrt3dTransformCallback, passing each callback a pointer to an Xrt3dTransformCallbackStruct structure. Unless the application sets Xrt3dTransformCallbackStruct's *doit* parameter to False, *ScaleUpdate()* applies the new scale to the widget.

SelectAxis()

Constrains rotation to a specified axis. This action takes one parameter that indicates which axis the user wishes to use:

X	Constrain rotation about the X-axis
Y	Constrain rotation about the Y-axis
Z	Constrain rotation about the Z-axis
Eye	Constrain rotation about the "eye" axis (the axis that extends through the center of the cube, perpendicular to the screen projection plane
None	Unconstrained rotation

TranslateUpdate()

Translates the graph. *TranslateUpdate()* calculates new values for the XtNxrt3dView[XY]Translate resources, and calls any callbacks specified by XtNxrt3dTransformCallback, passing each callback a pointer to an Xrt3dTransformCallbackStruct structure. Unless the application sets Xrt3dTransformCallbackStruct's *doit* parameter to False, *TranslateUpdate()* applies the new resource values to the widget.

ViewReset()

Restores all scaling and translations to XRT/3d defaults. *ViewReset()* calls any callbacks specified by XtNxrt3dTransformCallback, passing each callback a pointer to an Xrt3dTransformCallbackStruct structure containing the proposed values for the XtNxrt3dViewScale and XtNxrt3dView[XY]Translate resources. Unless the application sets Xrt3dTransformCallbackStruct's *doit* parameter to False, *ViewReset()* applies the new resource values to the widget.

ZoomUpdate()

Updates the "zoom rubberband".

ZoomEnd()

Zooms into the graph at the area selected by the "zoom rubberband". *ZoomEnd()* calculates new values for the `XtNxrt3dViewScale` and `XtNxrt3dView[XY]Translate` resources, and calls any callbacks specified by `XtNxrt3dTransformCallback`, passing each callback a pointer to an `Xrt3dTransformCallbackStruct` structure. Unless the application sets `Xrt3dTransformCallbackStruct`'s *doit* parameter to `False`, *ZoomEnd()* applies the new resource values to the widget.

ZoomCancel()

Cancels a zoom in progress, erasing the "zoom rubberband".

E

Data Types

This appendix lists the XRT/3d data types in alphabetical order. The C language definition of structures is also provided.

Xrt3dAdjust

Enumeration used by the `XtNxrt3dFooterAdjust` and `XtNxrt3dHeaderAdjust` resources:

```
XRT3D_ADJUST_LEFT
XRT3D_ADJUST_RIGHT
XRT3D_ADJUST_CENTER
```

Xrt3dAlign

Enumeration used with the `XtNxrt3dLegendOrientation` resource:

```
XRT3D_ALIGN_VERTICAL
XRT3D_ALIGN_HORIZONTAL
```

Xrt3dAnchor

Enumeration used with the `XtNxrt3dLegendAnchor` resource:

```
XRT3D_ANCHOR_NORTH
XRT3D_ANCHOR_SOUTH
XRT3D_ANCHOR_EAST
XRT3D_ANCHOR_WEST
XRT3D_ANCHOR_NORTHWEST
XRT3D_ANCHOR_NORTHEAST
XRT3D_ANCHOR_SOUTHWEST
XRT3D_ANCHOR_SOUTHEAST
```

Xrt3dAnnoMethod

Enumeration used to specify the `XtNxrt3d[XYZ]AnnoMethod`:

```
XRT3D_ANNO_VALUES
XRT3D_ANNO_DATA_LABELS
XRT3D_ANNO_VALUE_LABELS
```

Xrt3dAxis

Enumeration used to specify an axis.

```
XRT3D_AXIS_X
XRT3D_AXIS_Y
XRT3D_AXIS_Z
XRT3D_AXIS_NONE
XRT3D_AXIS_EYE
```

Xrt3dBarFormat

Enumeration used to specify the `XtNxrt3d[XY]BarFormat` (not applicable to scatter plots):

```
XRT3D_BAR_FIXED
XRT3D_BAR_HISTOGRAM
```

Xrt3dBorder

Enumeration used by the `XtNxrt3dHeaderBorder`, `XtNxrt3dFooterBorder`, `XtNxrt3dLegendBorder` and `XtNxrt3dGraphBorder` resources, and the text object's `XtNxrt3dTextBorder` resource:

```
XRT3D_BORDER_NONE
XRT3D_BORDER_PLAIN
XRT3D_BORDER_3D_IN
XRT3D_BORDER_3D_OUT
XRT3D_BORDER_ETCHED_IN
XRT3D_BORDER_ETCHED_OUT
XRT3D_BORDER_FRAME_IN
XRT3D_BORDER_FRAME_OUT
XRT3D_BORDER_BEVEL
XRT3D_BORDER_SHADOW
```

Xrt3dCallbackStruct

Structure used by callbacks defined with the `XtNxrt3dResizeCallback` and `XtNxrt3dExposeCallback` resources. See the resource descriptions in Appendix A for more information on each element.

```
typedef struct {
    int     reason;
    XEvent  *event;
    Window  window;
    Region  region;
    int     change;
} Xrt3dCallbackStruct;
```

Xrt3dColor

Platform specific type that is defined as *char* *.

Xrt3dContourStyle

The structure defining how a particular contour and zone are displayed (not applicable to scatter plots):

```
typedef struct {
    char            *fill_color;    /*for zoning*/
    char            *line_color;    /*line color*/
    int             line_width;     /*line width*/
    Xrt3dLinePattern lpat;          /*2D contours only*/

    unsigned long   res1;   /* reserved */
    unsigned long   res2;   /* reserved */
} Xrt3dContourStyle;
```

Xrt3dData

The structure defining data for use with `XtNxrt3dSurfaceData` and `XtNxrt3dZoneData`. It is a union of three data structures:

```
typedef union {
    Xrt3dGridData    g;
    Xrt3dIrGridData ig;
    Xrt3dPointData   p;
} Xrt3dData;
```

Xrt3dDataStyle

The structure used to define line and symbol style for each scatter plot series:

```
typedef struct {
    Xrt3dLineStyle      line_style;
    Xrt3dSymbolStyle    symbol_style;
} Xrt3dDataStyle;
```

Xrt3dDataType

Enumeration used for defining the type of an `Xrt3dData` structure:

```
XRT3D_DATA_GRID = 1
XRT3D_DATA_IRGRID
XRT3D_DATA_POINT
```

Xrt3dDistnMethod

Enumeration used for defining the distribution method used when DrawContours and/or DrawZones is `True`. Not applicable to scatter plots. There are currently two methods:

```
XRT3D_DISTN_LINEAR
XRT3D_DISTN_FROM_TABLE
```

Xrt3dDistnRange

Enumeration used for defining the distribution range of a distribution table. Not applicable to scatter plots. There are currently two methods:

```
XRT3D_DISTN_RANGE_DATA
XRT3D_DISTN_RANGE_ALL
```

Xrt3dDistnTable

The structure used to define custom distributions (not applicable to scatter plots):

```
typedef struct {
    int             nentries;
    double          *entry;
} Xrt3dDistnTable
```

Xrt3dDrawPSColor

Enumeration used for specifying color-handling with *Xrt3dDrawPS()*:

```
XRT3D_PS_COLOR_AS_IS
XRT3D_PS_COLOR_AUTO
XRT3D_PS_COLOR_MONO
```

Xrt3dGridData

The structure defining `Xrt3dData` when the type is `XRT3D_DATA_GRID`:

```
typedef struct {
    Xrt3dDataType   type; /*XRT3D_DATA_GRID*/
    int             numx, numy;
    double          noval;
    double          xstep, ystep;
    double          xorig, yorig;
    double          **values;
} Xrt3dGridData;
```

Xrt3dImage

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`.

Xrt3dImageLayout

Enumeration used by `XtNxrt3dImageLayout`, `XtNxrt3dGraphImageLayout`, `XtNxrt3dHeaderImageLayout`, `XtNxrt3dFooterImageLayout`, `XtNxrt3dLegendImageLayout`, and the text object's `XtNxrt3dTextImageLayout` resource:

```
XRT3D_IMAGELAYOUT_CENTERED = 1
XRT3D_IMAGELAYOUT_TILED
```

Xrt3dInterpMethod

Enumeration used to define the interpolation method in *Xrt3dDataWindow()* (not applicable to scatter plots):

```
XRT3D_INTERP_LINEAR_SPLINE
XRT3D_INTERP_CUBIC_SPLINE
```

Xrt3dIrGridData

The structure defining `Xrt3dData` when the type is `XRT3D_DATA_IRGRID`:

```
typedef struct {
    Xrt3dDataType   type; /*XRT3D_DATA_IRGRID*/
    int             numx, numy;
    double          noval;
    double          *xgrid, *ygrid;
    double          **values;
} Xrt3dIrGridData;
```

Xrt3dLegendStyle

Enumeration of various legend styles, used with the `XtNxrt3dLegendStyle` resource (not applicable to scatter plots):

```
XRT3D_LEGEND_STYLE_STEPPED
XRT3D_LEGEND_STYLE_CONTINUOUS
```

Xrt3dLinePattern

Enumeration of various line patterns, used within an `Xrt3dContourStyle` structure (applicable to contour lines, grid lines and scatter plot drop lines only):

```
XRT3D_LPAT_NONE
XRT3D_LPAT_SOLID
XRT3D_LPAT_LONG_DASH
XRT3D_LPAT_DOTTED
XRT3D_LPAT_SHORT_DASH
XRT3D_LPAT_LSL_DASH
XRT3D_LPAT_DASH_DOT
```

Xrt3dLineStyle

Structure that defines the line pattern, color and width pattern for lines (applicable to grid lines and scatter plot drop lines only):

```
typedef struct {
    Xrt3dLinePattern    pattern;
    Xrt3dColor          color;
    int                 width;
} Xrt3dLineStyle
```

Xrt3dMapCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrt3dMapCallback` list:

```
typedef struct {
    int             reason;     /* Read-only */
    XEvent          *event;     /* Read-only */
    String          *params;    /* Read-only */
    Cardinal        nparams;    /* Read-only */
    Xrt3dRegion     region;     /* Read-only */
    Xrt3dMapResult   map;       /* Read-only */
} Xrt3dMapCallbackStruct;
```

Xrt3dMapResult

Structure used to pass information about mapped pixel coordinates:

```
typedef struct {
    Position    pix_x, pix_y;
    double      x, y, z;
} Xrt3dMapResult;
```

Xrt3dModifyCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrt3dModifyCallback` list:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    String      *params;    /* Read-only */
    Cardinal    nparams;    /* Read-only */
    Boolean     doit;
} Xrt3dModifyCallbackStruct;
```

Xrt3dPickCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrt3dPickCallback` list:

```
typedef struct {
    int             reason;    /* Read-only */
    XEvent          *event;    /* Read-only */
    String          *params;   /* Read-only */
    Cardinal        nparams;   /* Read-only */
    Xrt3dRegion     region;    /* Read-only */
    Xrt3dPickResult pick;      /* Read-only */
} Xrt3dPickCallbackStruct;
```

Xrt3dPickResult

Structure used to pass information about picked pixel coordinates:

```
typedef struct {
    Position    pix_x, pix_y;
    int         xindex, yindex;
    int         distance;
} Xrt3dPickResult;
```

Xrt3dPoint3D

Point structure for scatter plots.

```
typedef struct {
    double      x, y, z;
}
```

Xrt3dPointData

The structure defining `Xrt3dData` when the type is `XRT3D_DATA_POINT`:

```
typedef struct {
    Xrt3dDataType       type;
    int                 nseries;
    double              noval;
    Xrt3dPointSeries    *series;
} Xrt3dPointData;
```

Xrt3dPointSeries

Structure for scatter plots that holds a list of points in a series:

```
typedef struct {
    int             npoints;
    Xrt3dPoint3D    *npoints;
} Xrt3dPointSeries;
```

Xrt3dPreviewMethod

Enumeration used to specify how interactive rotations are previewed to the user:

```
XRT3D_PREVIEW_CUBE
XRT3D_PREVIEW_FULL
```

Xrt3dProperty

Enumeration of cross-platform names for XRT/3d resources, used with the *Xrt3dGetPropString()*, *Xrt3dGetValues()*, *Xrt3dSetPropString()*, *Xrt3dSetValues()*, *Xrt3dTextGetPropString()*, *Xrt3dTextGetValues()*, *Xrt3dTextSetValues()*, and *Xrt3dTextSetPropString()* procedures. For a list of the definition for each resource, see page 205.

Xrt3dReason

Enumeration used to define the reason for a callback. Used with the `XtNxrt3dMapCallback`, `XtNxrt3dModifyCallback`, `XtNxrt3dPickCallback`, `XtNxrt3dRotateCallback`, and `XtNxrt3dTransformCallback` resources:

```
XRT3D_REASON_MODIFY_START
XRT3D_REASON_MODIFY_END
XRT3D_REASON_ROTATE
XRT3D_REASON_TRANSFORM
XRT3D_REASON_MAP
XRT3D_REASON_PICK
```

Xrt3dRegion

Enumeration of map or pick results. Returned by *Xrt3dMap()* and *Xrt3dPick()*:

```
XRT3D_RGN_NOWHERE
XRT3D_RGN_IN_GRAPH
XRT3D_RGN_IN_LEGEND
XRT3D_RGN_IN_FOOTER
XRT3D_RGN_IN_HEADER
```

Xrt3dRotateCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrt3dRotateCallback` list:

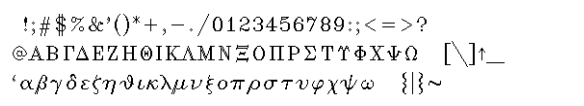
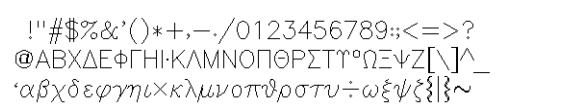
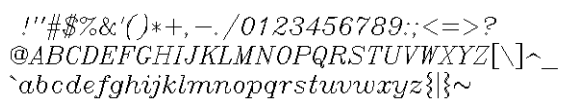
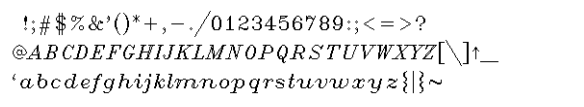
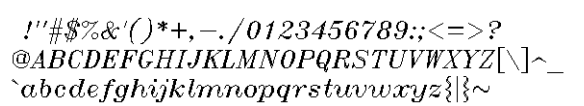
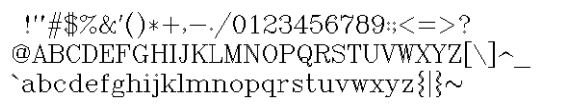
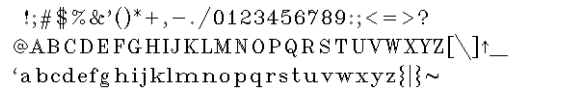
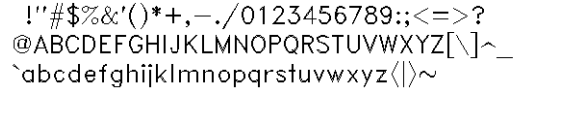
```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    String      *params;    /* Read-only */
    Cardinal    nparams;    /* Read-only */
    double      xrotation;
    double      yrotation;
    double      zrotation;
    Boolean     doit;
} Xrt3dRotateCallbackStruct;
```

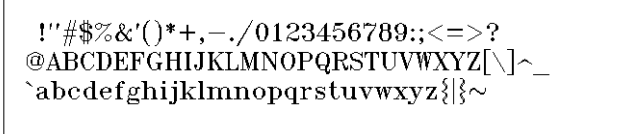
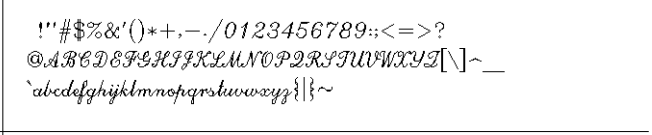
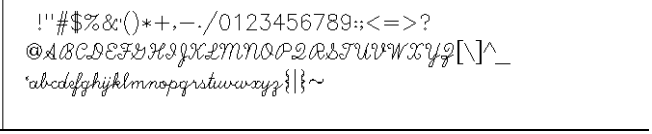
Xrt3dStrokeFont

Enumeration of stroke font names. Used by `XtNxrt3dAxisStrokeFont` and `XtNxrt3dAxisTitleStrokeFont`:

```
XRT3D_SF_CYRILLIC_COMPLEX
XRT3D_SF_GOTHIC_ENGLISH
XRT3D_SF_GOTHIC_GERMAN
XRT3D_SF_GOTHIC_ITALIAN
XRT3D_SF_GREEK_COMPLEX
XRT3D_SF_GREEK_COMPLEX_SMALL
XRT3D_SF_GREEK_SIMPLEX
XRT3D_SF_ITALIC_COMPLEX
XRT3D_SF_ITALIC_COMPLEX_SMALL
XRT3D_SF_ITALIC_TRIPLEX
XRT3D_SF_ROMAN_COMPLEX
XRT3D_SF_ROMAN_COMPLEX_SMALL
XRT3D_SF_ROMAN_DUPLEX
XRT3D_SF_ROMAN_SIMPLEX
XRT3D_SF_ROMAN_TRIPLEX
XRT3D_SF_SCRIPT_COMPLEX
XRT3D_SF_SCRIPT_SIMPLEX
```

Stroke Font Name	Glyphs
Cyrillic Complex	!"#Ыцы'()*+,-./0123456789:;<=>? @АБЭДИФГЖИЧКЛМНОПШРСТЮВЩХУЗЕ\ЪЯЬ ЦабЭдйфгжичклмнопшрстювщхузеъяь
Gothic English	!"#$%&'()*+,-./0123456789:;<=>? @𝔄𝔅ℭ𝔇𝔈𝔉𝔊ℌℑ𝔍𝔎𝔏𝔐𝔑𝔒𝔓𝔔ℜ𝔖𝔗𝔘𝔙𝔚𝔛𝔜ℨ[\]^_ `𝔞𝔟𝔠𝔡𝔢𝔣𝔤𝔥𝔦𝔧𝔨𝔩𝔪𝔫𝔬𝔭𝔮𝔯𝔰𝔱𝔲𝔳𝔴𝔵𝔶𝔷{|}~
Gothic German	!"#$%&'()*+,-./0123456789:;<=>? @𝕬𝕭𝕮𝕯𝕰𝕱𝕲𝕳𝕴𝕵𝕶𝕷𝕸𝕹𝕺𝕻𝕼𝕽𝕾𝕿𝖀𝖁𝖂𝖃𝖄𝖅[\]^_ `𝖆𝖇𝖈𝖉𝖊𝖋𝖌𝖍𝖎𝖏𝖐𝖑𝖒𝖓𝖔𝖕𝖖𝖗𝖘𝖙𝖚𝖛𝖜𝖝𝖞𝖟{|}~
Gothic Italian	!"#$%&'()*+,-./0123456789:;<=>? @𝐴𝐵𝐶𝐷𝐸𝐹𝐺𝐻𝐼𝐽𝐾𝐿𝑀𝑁𝑂𝑃𝑄𝑅𝑆𝑇𝑈𝑉𝑊𝑋𝑌𝑍[\]^_ `𝑎𝑏𝑐𝑑𝑒𝑓𝑔ℎ𝑖𝑗𝑘𝑙𝑚𝑛𝑜𝑝𝑞𝑟𝑠𝑡𝑢𝑣𝑤𝑥𝑦𝑧{|}~
Greek Complex	!"#$%&'()*+,-./0123456789:;<=>? @ΑΒΧΔΕΦΓΗΙ·ΚΛΜΝΟΠΘΡΣΤΥ°ΩΞΨΖ[\]^_ `αβχδεφγηι×κλμνοπϑρστυ÷ωξψζ{|}~

Stroke Font Name	Glyphs
Greek Complex Small	
Greek Simplex	
Italic Complex	
Italic Complex Small	
Italic Triplex	
Roman Complex	
Roman Complex Small	
Roman Duplex	

Stroke Font Name	Glyphs
Roman Simplex	
Roman Triplex	
Script Complex	
Script Simplex	

Xrt3dSymbolPattern

Enumeration for symbol patterns used for points of scatter plot.

```
XRT3D_SYMBOL_NONE
XRT3D_SYMBOL_DOT
XRT3D_SYMBOL_BOX
XRT3D_SYMBOL_TRI
XRT3D_SYMBOL_DIAMOND
XRT3D_SYMBOL_STAR
XRT3D_SYMBOL_VERT_LINE
XRT3D_SYMBOL_HORIZ_LINE
XRT3D_SYMBOL_CROSS
XRT3D_SYMBOL_CIRCLE
XRT3D_SYMBOL_SQUARE
XRT3D_SYMBOL_INVERT_TRI
XRT3D_SYMBOL_DIAG_CROSS
XRT3D_SYMBOL_OPEN_TRI
XRT3D_SYMBOL_OPEN_DIAMOND
XRT3D_SYMBOL_OPEN_INVERT_TRI
```

Xrt3dSymbolStyle

Structure that defines the symbol color, size and pattern for a given data series (scatter plots only).

```
typedef struct {
    Xrt3dSymbolPattern  pattern;
    Xrt3dColor          color;
    int                 size;
} Xrt3dSymbolStyle
```

Xrt3dTextAttachMethod

Enumeration used to specify how XRT/3d text objects are attached to the graph:

```
XRT3D_ATTACH_INDEX
XRT3D_ATTACH_PIXEL
XRT3D_ATTACH_POINT
```

Xrt3dTransformCallbackStruct

Structure that defines the information passed to callbacks on the `XtNxrt3dTransformCallback` list:

```
typedef struct {
    int         reason;     /* Read-only */
    XEvent      *event;     /* Read-only */
    String      *params;    /* Read-only */
    Cardinal    nparams;    /* Read-only */
    double      scale;
    double      xtranslate;
    double      ytranslate;
    Boolean     doit;
} Xrt3dTransformCallbackStruct;
```

Xrt3dType

Enumeration used to specify the `XtNxrt3dType`:

```
XRT3D_TYPE_SURFACE
XRT3D_TYPE_BAR
XRT3D_TYPE_SCATTER
```

Xrt3dValueLabel

Structure used to supply value labels for an axis:

```
typedef struct {
    double      value;
    char        *label;
} Xrt3dValueLabel;
```

Xrt3dXYColor

Structure used to specify coloring of individual bars (not applicable to scatter plots).

```
typedef struct {
    int             xindex;
    int             yindex;
    char *          color;
    unsigned long   res;        /* reserved */
} Xrt3dXYColor;
```

Xrt3dZoneMethod

Enumeration used to specify the zoning method. Not applicable to scatter plots. Used with `XtNxrt3dZoneMethod`:

```
XRT3D_ZONE_CONTOURS
XRT3D_ZONE_CELLS
```

F

Resource Names, Types and Classes

This appendix lists the resource type and resource class for each XRT/3d resource. Underneath each resource name is listed its `Xrt3dProperty` enumeration.

Name	Class	Type
XtNxrt3dAxisStrokeFont XRT3D_AXIS_STROKE_FONT	XtCXrt3dStrokeFont	XtRXrt3dStrokeFont
XtNxrt3dAxisStrokeSize XRT3D_AXIS_STROKE_SIZE	XtCXrt3dStrokeFontSize	XtRDimension
XtNxrt3dAxisTitleStrokeFont XRT3D_AXIS_TITLE_STROKE_FONT	XtCXrt3dStrokeFont	XtRXrt3dStrokeFont
XtNxrt3dAxisTitleStrokeSize XRT3D_AXIS_TITLE_STROKE_SIZE	XtCXrt3dStrokeFontSize	XtRDimension
XtNxrt3dBackgroundColor XRT3D_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtRXrt3dColor
XtNxrt3dColorFunc XRT3D_COLOR_FUNC	XtCXrt3dColorFunc	XtRFunction
XtNxrt3dContourStyles XRT3D_CONTOUR_STYLES	XtCXrt3dContourStyles	XtRXrt3dContourStyles
XtNxrt3dDataAreaBackgroundColor XRT3D_DATA_AREA_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtR3dColor
XtNxrt3dDataStyles XRT3D_DATA_STYLES	XtCXrt3dDataStyles	XtRXrt3dDataStyles
XtNxrt3dDataStylesUseDefault XRT3D_DATA_STYLES_USE_DEFAULT	XtCXrt3dDataStylesUseDefault	XtRBoolean
XtNxrt3dDebug XRT3D_DEBUG	XtCXrt3dDebug	XtRBoolean
XtNxrt3dDistnMethod XRT3D_DISTN_METHOD	XtCXrt3dDistnMethod	XtRXrt3dDistnMethod

Name	Class	Type
XtNxrt3dDistnTable XRT3D_DISTN_TABLE	XtCXrt3dDistnTable	XtRXrt3dDistnTable
XtNxrt3dDoubleBuffer XRT3D_DOUBLE_BUFFER	XtCXrt3dDoubleBuffer	XtRBoolean
XtNxrt3dDrawContours XRT3D_DRAW_CONTOURS	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dDrawDropLines XRT3D_DRAW_DROPLINES	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dDrawHiddenLines XRT3D_DRAW_HIDDEN_LINES	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dDrawMesh XRT3D_DRAW_MESH	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dDrawShaded XRT3D_DRAW_SHADED	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dDrawZones XRT3D_DRAW_ZONES	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dExposeCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dFooterAdjust XRT3D_FOOTER_ADJUST	XtCXrt3dAdjust	XtRXrt3dAdjust
XtNxrt3dFooterBackgroundColor XRT3D_FOOTER_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtRXrt3dColor
XtNxrt3dFooterBorder XRT3D_FOOTER_BORDER	XtCXrt3dBorder	XtRXrt3dBorder
XtNxrt3dFooterBorderWidth XRT3D_FOOTER_BORDER_WIDTH	XtCXrt3dBorderWidth	XtRDimension
XtNxrt3dFooterFont XRT3D_FOOTER_FONT	XmCFontList	XmRFontList
XtNxrt3dFooterForegroundColor XRT3D_FOOTER_FOREGROUND_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dFooterHeight XRT3D_FOOTER_HEIGHT	XtCXrt3dDimension	XtRDimension
XtNxrt3dFooterImage XRT3D_FOOTER_IMAGE	XtCXrt3dImage	XtRXrt3dImage
XtNxrt3dFooterImageLayout XRT3D_FOOTER_IMAGE_LAYOUT	XtCXrt3dImageLayout	XtRXrt3dImageLayout
XtNxrt3dFooterImageMinimumSize XRT3D_FOOTER_IMAGE_MINIMUM_SIZE	XtCXrt3dImageMinimumSize	XtRBoolean
XtNxrt3dFooterStrings XRT3D_FOOTER_STRINGS	XtCXrt3dStrings	XtRXrt3dStrings
XtNxrt3dFooterWidth XRT3D_FOOTER_WIDTH	XtCXrt3dDimension	XtRDimension

Name	Class	Type
XtNxrt3dFooterX XRT3D_FOOTER_X	XtCXrt3dPosition	XtRPosition
XtNxrt3dFooterXUseDefault XRT3D_FOOTER_X_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dFooterY XRT3D_FOOTER_Y	XtCXrt3dPosition	XtRPosition
XtNxrt3dFooterYUseDefault XRT3D_FOOTER_Y_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dForegroundColor XRT3D_FOREGROUND_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dGraphBackgroundColor XRT3D_GRAPH_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtRXrt3dColor
XtNxrt3dGraphBorder XRT3D_GRAPH_BORDER	XtCXrt3dBorder	XtRXrt3dBorder
XtNxrt3dGraphBorderWidth XRT3D_GRAPH_BORDER_WIDTH	XtCXrt3dBorderWidth	XtRDimension
XtNxrt3dGraphForegroundColor XRT3D_GRAPH_FOREGROUND_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dGraphHeight XRT3D_GRAPH_HEIGHT	XtCXrt3dDimension	XtRDimension
XtNxrt3dGraphHeightUseDefault XRT3D_GRAPH_HEIGHT_USE_DEFAULT	XtCXrt3dDimensionUseDefault	XtRBoolean
XtNxrt3dGraphImage XRT3D_GRAPH_IMAGE	XtCXrt3dImage	XtRXrt3dImage
XtNxrt3dGraphImageLayout XRT3D_GRAPH_IMAGE_LAYOUT	XtCXrt3dImageLayout	XtRXrt3dImageLayout
XtNxrt3dGraphWidth XRT3D_GRAPH_WIDTH	XtCXrt3dDimension	XtRDimension
XtNxrt3dGraphWidthUseDefault XRT3D_GRAPH_WIDTH_USE_DEFAULT	XtCXrt3dDimensionUseDefault	XtRBoolean
XtNxrt3dGraphX XRT3D_GRAPH_X	XtCXrt3dPosition	XtRPosition
XtNxrt3dGraphXUseDefault XRT3D_GRAPH_X_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dGraphY XRT3D_GRAPH_Y	XtCXrt3dPosition	XtRPosition
XtNxrt3dGraphYUseDefault XRT3D_GRAPH_Y_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dHeaderAdjust XRT3D_HEADER_ADJUST	XtCXrt3dAdjust	XtRXrt3dAdjust

Name	Class	Type
XtNxrt3dHeaderBackgroundColor XRT3D_HEADER_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtRXrt3dColor
XtNxrt3dHeaderBorder XRT3D_HEADER_BORDER	XtCXrt3dBorder	XtRXrt3dBorder
XtNxrt3dHeaderBorderWidth XRT3D_HEADER_BORDER_WIDTH	XtCXrt3dBorderWidth	XtRDimension
XtNxrt3dHeaderFont XRT3D_HEADER_FONT	XmCFontList	XmRFontList
XtNxrt3dHeaderForegroundColor XRT3D_HEADER_FOREGROUND_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dHeaderHeight XRT3D_HEADER_HEIGHT	XtCXrt3dDimension	XtRDimension
XtNxrt3dHeaderImage XRT3D_HEADER_IMAGE	XtCXrt3dImage	XtRXrt3dImage
XtNxrt3dHeaderImageLayout XRT3D_HEADER_IMAGE_LAYOUT	XtCXrt3dImageLayout	XtRXrt3dImageLayout
XtNxrt3dHeaderImageMinimumSize XRT3D_HEADER_IMAGE_MINIMUM_SIZE	XtCXrt3dImageMinimumSize	XtRBoolean
XtNxrt3dHeaderStrings XRT3D_HEADER_STRINGS	XtCXrt3dStrings	XtRXrt3dStrings
XtNxrt3dHeaderWidth XRT3D_HEADER_WIDTH	XtCXrt3dDimension	XtRDimension
XtNxrt3dHeaderX XRT3D_HEADER_X	XtCXrt3dPosition	XtRPosition
XtNxrt3dHeaderXUseDefault XRT3D_HEADER_X_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dHeaderY XRT3D_HEADER_Y	XtCXrt3dPosition	XtRPosition
XtNxrtHeaderYUseDefault XRT3D_HEADER_Y_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dImage XRT3D_IMAGE	XtCXrt3dImage	XtRXrt3dImage
XtNxrt3dImageLayout XRT3D_IMAGE_LAYOUT	XtCXrt3dImageLayout	XtRXrt3dImageLayout
XtNxrt3dLegendAnchor XRT3D_LEGEND_ANCHOR	XtCXrt3dAnchor	XtRXrt3dAnchor
XtNxrt3dLegendBackgroundColor XRT3D_LEGEND_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtRXrt3dColor
XtNxrt3dLegendBorder XRT3D_LEGEND_BORDER	XtCXrt3dBorder	XtRXrt3dBorder

Name	Class	Type
XtNxrt3dLegendBorderWidth XRT3D_LEGEND_BORDER_WIDTH	XtCXrt3dBorderWidth	XtRDimension
XtNxrt3dLegendDistnRange XRT3D_LEGEND_DISTN_RANGE	XtCXrt3dLegendDistnRange	XtRXrt3dDistnRange
XtNxrt3dLegendFont XRT3D_LEGEND_FONT	XmCFontList	XmRFontList
XtNxrt3dLegendForegroundColor XRT3D_LEGEND_FOREGROUND_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dLegendHeight XRT3D_LEGEND_HEIGHT	XtCXrt3dDimension	XtRDimension
XtNxrt3dLegendImage XRT3D_LEGEND_IMAGE	XtCXrt3dImage	XtRXrt3dImage
XtNxrt3dLegendImageLayout XRT3D_LEGEND_IMAGE_LAYOUT	XtCXrt3dImageLayout	XtRXrt3dImageLayout
XtNxrt3dLegendLabelFilterc XRT3D_LEGEND_LABEL_FILTER	XtCXrt3dLegendLabelFilter	XtRInt
XtNxrt3dLegendLabelFunc XRT3D_LEGEND_LABEL_FUNC	XtCXrt3dLegendLabelFunc	XtRFunction
XtNxrt3dLegendOrientation XRT3D_LEGEND_ORIENTATION	XtCXrt3dOrientation	XtRXrt3dAlign
XtNxrt3dLegendShow XRT3D_LEGEND_SHOW	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dLegendStrings XRT3D_LEGEND_STRINGS	XtCXrt3dStrings	XtRXrt3dStrings
XtNxrt3dLegendStyle XRT3D_LEGEND_STYLE	XtCXrt3dLegendStyle	XtRXrt3dLegendStyle
XtNxrt3dLegendTitle XRT3D_LEGEND_TITLE	XtCXrt3dLegendTitle	XtRXrt3dString
XtNxrt3dLegendWidth XRT3D_LEGEND_WIDTH	XtCXrt3dDimension	XtRDimension
XtNxrt3dLegendX XRT3D_LEGEND_X	XtCXrt3dPosition	XtRPosition
XtNxrt3dLegendXUseDefault XRT3D_LEGEND_X_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dLegendY XRT3D_LEGEND_Y	XtCXrt3dPosition	XtRPosition
XtNxrt3dLegendYUseDefault XRT3D_LEGEND_Y_USE_DEFAULT	XtCXrt3dPositionUseDefault	XtRBoolean
XtNxrt3dMapCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dMeshBottomColor XRT3D_MESH_BOTTOM_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor

Name	Class	Type
XtNxrt3dMeshTopColor XRT3D_MESH_TOP_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dModifyCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dNumDistnLevels XRT3D_NUM_DISTN_LEVELS	XtCXrt3dNumDistnLevels	XtRInt
XtNxrt3dPerspectiveDepth XRT3D_PERSPECTIVE_DEPTH	XtCXrt3dPerspectiveDepth	XtRXrt3dDouble
XtNxrt3dPickCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dPreviewMethod XRT3D_PREVIEW_METHOD	XtCXrt3dPreviewMethod	XtRXrt3dPreviewMethod
XtNxrt3dProjectZMax XRT3D_PROJECT_ZMAX	XtCXrt3dProject	XtRXrt3dProject
XtNxrt3dProjectZMin XRT3D_PROJECT_ZMIN	XtCXrt3dProject	XtRXrt3dProject
XtNxrt3dRepaint XRT3D_REPAINT	XtCXrt3dRepaint	XtRBoolean
XtNxrt3dResizeCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dRotateCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dSolidSurface XRT3D_SOLID_SURFACE	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dSurfaceBottomColor XRT3D_SURFACE_BOTTOM_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dSurfaceData XRT3D_SURFACE_DATA	XtCXrt3dData	XtRXrt3dData
XtNxrt3dSurfaceTopColor XRT3D_SURFACE_TOP_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dTransformCallback [a]	XtCXrt3dCallback	XtRCallback
XtNxrt3dType XRT3D_TYPE	XtCXrt3dType	XtRXrt3dType
XtNxrt3dViewNormalized XRT3D_VIEW_NORMALIZED	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dViewScale XRT3D_VIEW_SCALE	XtCXrt3dViewControl	XtRXrt3dDouble
XtNxrt3dViewXTranslate XRT3D_VIEW_XTRANSLATE	XtCXrt3dViewControl	XtRXrt3dDouble
XtNxrt3dViewYTranslate XRT3D_VIEW_YTRANSLATE	XtCXrt3dViewControl	XtRXrt3dDouble
XtNxrt3dXAnnoMethod XRT3D_XANNO_METHOD	XtCXrt3dAnnoMethod	XtRXrt3dAnnoMethod

Name	Class	Type
XtNxrt3dXAxisShow XRT3D_XAXIS_SHOW	XtCXrt3dAxisShow	XtRBoolean
XtNxrt3dXAxisTitle XRT3D_XAXIS_TITLE	XtCXrt3dAxisTitle	XtRXrt3dString
XtNxrt3dXBarFormat XRT3D_XBAR_FORMAT	XtCXrt3dBarFormat	XtRXrt3dBarFormat
XtNxrt3dXBarSpacing XRT3D_XBAR_SPACING	XtCXrt3dBarSpacing	XtRXrt3dDouble
XtNxrt3dXDataLabels XRT3D_XDATA_LABELS	XtCXrt3dDataLabels	XtRXrt3dStrings
XtNxrt3dXGridLines XRT3D_XGRID_LINES	XtCXrt3dGridLines	XtRXrt3dGridLines
XtNxrt3dXGridLineStyle XRT3D_XGRID_LINE_STYLE	XtCXrt3dGridLineStyle	XtRXrt3dLineStyle
XtNxrt3dXGridLineStyleUseDefault XRT3D_XGRID_LINE_STYLE_USE_DEFAULT	XtCXrt3dGridLineStyleUseDefault	XtRBoolean
XtNxrt3dXMax XRT3D_XMAX	XtCXrt3dAxisBounds	XtR3dDouble
XtNxrt3dXMaxUseDefault XRT3D_XMAX_USE_DEFAULT	XtCXrt3dAxisBoundsUseDefault	XtRBoolean
XtNxrt3dXMeshFilter XRT3D_XMESH_FILTER	XtCXrt3dMeshFilter	XtRInt
XtNxrt3dXMeshShow XRT3D_XMESH_SHOW	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dXMin XRT3D_XMIN	XtCXrt3dAxisBounds	XtRXrt3dDouble
XtNxrt3dXMinUseDefault XRT3D_XMIN_USE_DEFAULT	XtCXrt3dAxisBoundsUseDefault	XtRBoolean
XtNxrt3dXRotation XRT3D_XROTATION	XtCXrt3dRotation	XtRXrt3dDouble
XtNxrt3dXScale XRT3D_XSCALE	XtCXrt3dScale	XtRXrt3dDouble
XtNxrt3dXValueLabels XRT3D_XVALUE_LABELS	XtCXrt3dValueLabels	XtRXrt3dValueLabels
XtNxrt3dXYColors XRT3D_XY_COLORS	XtCXrt3dXYColors	XtRXrt3dXYColors
XtNxrt3dYAnnoMethod XRT3D_YANNO_METHOD	XtCXrt3dAnnoMethod	XtRXrt3dAnnoMethod
XtNxrt3dYAxisShow XRT3D_YAXIS_SHOW	XtCXrt3dAxisShow	XtRBoolean

Name	Class	Type
XtNxrt3dYAxisTitle XRT3D_YAXIS_TITLE	XtCXrt3dAxisTitle	XtRXrt3dString
XtNxrt3dYBarFormat XRT3D_YBAR_FORMAT	XtCXrt3dBarFormat	XtRXrt3dBarFormat
XtNxrt3dYBarSpacing XRT3D_YBAR_SPACING	XtCXrt3dBarSpacing	XtRXrt3dDouble
XtNxrt3dYDataLabels XRT3D_YDATA_LABELS	XtCXrt3dDataLabels	XtRXrt3dStrings
XtNxrt3dYGridLines XRT3D_YGRID_LINES	XtCXrt3dGridLines	XtRXrt3dGridLines
XtNxrt3dYGridLineStyle XRT3D_YGRID_LINE_STYLE	XtCXrt3dGridLineStyle	XtRXrt3dLineStyle
XtNxrt3dYGridLineStyleUseDefault XRT3D_YGRID_LINE_STYLE_USE_DEFAULT	XtCXrt3dGridLineStyleUseDefault	XtRBoolean
XtNxrt3dYMax XRT3D_YMAX	XtCXrt3dAxisBounds	XtRXrt3dDouble
XtNxrt3dYMaxUseDefault XRT3D_YMAX_USE_DEFAULT	XtCXrt3dAxisBoundsUseDefault	XtRBoolean
XtNxrt3dYMeshFilter XRT3D_YMESH_FILTER	XtCXrt3dMeshFilter	XtRInt
XtNxrt3dYMeshShow XRT3D_YMESH_SHOW	XtCXrt3dDrawOption	XtRBoolean
XtNxrt3dYMin XRT3D_YMIN	XtCXrt3dAxisBounds	XtRXrt3dDouble
XtNxrt3dYMinUseDefault XRT3D_YMIN_USE_DEFAULT	XtCXrt3dAxisBoundsUseDefault	XtRBoolean
XtNxrt3dYRotation XRT3D_YROTATION	XtCXrt3dRotation	XtRXrt3dDouble
XtNxrt3dYScale XRT3D_YSCALE	XtCXrt3dScale	XtRXrt3dDouble
XtNxrt3dYValueLabels XRT3D_YVALUE_LABELS	XtCXrt3dValueLabels	XtRXrt3dValueLabels
XtNxrt3dZAnnoMethod XRT3D_ZANNO_METHOD	XtCXrt3dAnnoMethod	XtRXrt3dAnnoMethod
XtNxrt3dZAxisShow XRT3D_ZAXIS_SHOW	XtCXrt3dAxisShow	XtRBoolean
XtNxrt3dZAxisTitle XRT3D_ZAXIS_TITLE	XtCXrt3dAxisTitle	XtRString
XtNxrt3dZGridLines XRT3D_ZGRID_LINES	XtCXrt3dGridLines	XtRXrt3dGridLines

Name	Class	Type
XtNxrt3dZGridLineStyle XRT3D_ZGRID_LINE_STYLE	XtCXrt3dGridLineStyle	XtRXrt3dLineStyle
XtNxrt3dZGridLineStyleUseDefault XRT3D_ZGRID_LINE_STYLE_USE_DEFAULT	XtCXrt3dGridLineStyleUseDefault	XtRBoolean
XtNxrt3dZMax XRT3D_ZMAX	XtCXrt3dAxisBounds	XtRXrt3dDouble
XtNxrt3dZMaxUseDefault XRT3D_ZMAX_USE_DEFAULT	XtCXrt3dAxisBoundsUseDefault	XtRBoolean
XtNxrt3dZMin XRT3D_ZMIN	XtCXrt3dAxisBounds	XtRXrt3dDouble
XtNxrt3dZMinUseDefault XRT3D_ZMIN_USE_DEFAULT	XtCXrt3dAxisBoundsUseDefault	XtRBoolean
XtNxrt3dZoneData XRT3D_ZONE_DATA	XtCXrt3dData	XtRXrt3dData
XtNxrt3dZoneMethod XRT3D_ZONE_METHOD	XtCXrt3dZoneMethod	XtRXrt3dZoneMethod
XtNxrt3dZOrigin XRT3D_ZORIGIN	XtCXrt3dOrigin	XtRXrt3dDouble
XtNxrt3dZRotation XRT3D_ZROTATION	XtCXrt3dRotation	XtRXrt3dDouble
XtNxrt3dZScale XRT3D_ZSCALE	XtCXrt3dScale	XtRXrt3dDouble
XtNxrt3dZValueLabels XRT3D_ZVALUE_LABELS	XtCXrt3dValueLabels	XtRXrt3dValueLabels

a. This resource is not used on Windows so no cross-platform name is defined.

XRT/3d Text Object Resources

Name	Class	Type
XtNxrt3dTextAdjust XRT3D_TEXT_ADJUST	XtCXrt3dAdjust	XtRXrt3dAdjust
XtNxrt3dTextAttachIndexX XRT3D_TEXT_ATTACH_INDEX_X	XtCXrt3dTextAttachIndex	XtRInt
XtNxrt3dTextAttachIndexY XRT3D_TEXT_ATTACH_INDEX_Y	XtCXrt3dTextAttachIndex	XtRInt
XtNxrt3dTextAttachIndexPoint XRT3D_TEXT_ATTACH_INDEX_POINT	XtCXrt3dTextAttachIndex	XtRInt
XtNxrt3dTextAttachIndexSeries XRT3D_TEXT_ATTACH_INDEX_SERIES	XtCXrt3dTextAttachIndex	XtRInt

XRT/3d Text Object Resources

Name	Class	Type
XtNxrt3dTextAttachMethod XRT3D_TEXT_ATTACH_METHOD	XtCXrt3dTextAttachMethod	XtRXrt3dAttachMethod
XtNxrt3dTextAttachPixelX XRT3D_TEXT_ATTACH_PIXEL_X	XtCXrt3dTextAttachPixel	XtRPosition
XtNxrt3dTextAttachPixelY XRT3D_TEXT_ATTACH_PIXEL_Y	XtCXrt3dTextAttachPixel	XtRPosition
XtNxrt3dTextAttachPointX XRT3D_TEXT_ATTACH_POINT_X	XtCXrt3dTextAttachPoint	XtRXrt3dDouble
XtNxrt3dTextAttachPointY XRT3D_TEXT_ATTACH_POINT_Y	XtCXrt3dTextAttachPoint	XtRXrt3dDouble
XtNxrt3dTextAttachPointZ XRT3D_TEXT_ATTACH_POINT_Z	XtCXrt3dTextAttachPoint	XtRXrt3dDouble
XtNxrt3dTextBackgroundColor XRT3D_TEXT_BACKGROUND_COLOR	XtCXrt3dBackgroundColor	XtRXrt3dColor
XtNxrt3dTextBorder XRT3D_TEXT_BORDER	XtCXrt3dBorder	XtRXrt3dBorder
XtNxrt3dTextBorderWidth XRT3D_TEXT_BORDER_WIDTH	XtCXrt3dBorderWidth	XtRDimension
XtNxrt3dTextFont XRT3D_TEXT_FONT	XmCFontList	XmRFontList
XtNxrt3dTextForegroundColor XRT3D_TEXT_FOREGROUND_COLOR	XtCXrt3dForegroundColor	XtRXrt3dColor
XtNxrt3dTextImage XRT3D_TEXT_IMAGE	XtCXrt3dImage	XtRXrt3dImage
XtNxrt3dTextImageLayout XRT3D_TEXT_IMAGE_LAYOUT	XtCXrt3dImageLayout	XtRXrt3dImageLayout
XtNxrt3dTextImageMinimumSize XRT3D_TEXT_IMAGE_MINIMUM_SIZE	XtCXrt3dImageMinimumSize	XtRBoolean
XtNxrt3dTextLineShow XRT3D_TEXT_LINE_SHOW	XtCXrt3dTextDrawOptions	XtRBoolean
XtNxrt3dTextList XRT3D_TEXT_LIST	XtCXrt3dTextList	XtRXrt3dTextList
XtNxrt3dTextOffsetX XRT3D_TEXT_OFFSET_X	XtCXrt3dPosition	XtRPosition
XtNxrt3dTextOffsetY XRT3D_TEXT_OFFSET_Y	XtCXrt3dPosition	XtRPosition
XtNxrt3dTextPlane XRT3D_TEXT_PLANE	XtCXrt3dTextPlane	XtRXrt3dPlane
XtNxrt3dTextPrintFont XRT3D_TEXT_PRINT_FONT	XtCString	XtRXrt3dString

XRT/3d Text Object Resources

Name	Class	Type
XtNxrt3dTextShow XRT3D_TEXT_SHOW	XtCXrt3dTextDrawOptions	XtRBoolean
XtNxrt3dTextStrings XRT3D_TEXT_STRINGS	XtCXrt3dStrings	XtRXrt3dStrings
XtNxrt3dTextStrokeFont XRT3D_TEXT_STROKE_FONT	XtCXrt3dStrokeFont	XtRXrt3dStrokeFont
XtNxrt3dTextStrokeSize XRT3D_TEXT_STROKE_SIZE	XtCXrt3dStrokeFontSize	XtRDimension

G

Resource File and GUI Builder Usage

Syntax
Example Resource File

This appendix lists the syntax for setting XRT/3d resources in resource files, using GUI builder tools, and using *Xrt3dSetPropString()*. It also lists a sample resource file. XRT/3d only defines resource converters for the data types not already defined by Xt and Motif.

All XRT/3d resources except `UseDefault` resources may be defined in a resource file. All XRT/3d text object resources may be defined in a resource file.

G.1 Syntax

Comments
Any text after an exclamation mark (`!`) on a line is ignored.

Continuation
Lines are continued on subsequent lines in the file by making a backslash (`\`) the last character in the line.

Booleans
"`True`" or "`False`" in upper, lower, or mixed case.

Enumerated Types
All resources with enumerated types use a consistent naming convention based on the enumerated values. For all resource types, the leading `XRT3D_` 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 `XRT3D_BAR_FIXED` to be used for `XtNxrt3dXBarFormat`:

```
*.XtXrt3d.xrt3dXBarFormat: BarFixed
*.XtXrt3d.xrt3dXBarFormat: BARFIXED
*.XtXrt3d.xrt3dXBarFormat: barfixed
```

Data

Resources of type `XtRXrt3dData` are specified with any valid filename. The resource converter opens the file and calls *Xrt3dMakeDataFromFile()*. Environment variables are allowed. Any error messages are written to *stderr*. An example:

```
*.XtXrt3d.xrt3dData: $XRTHOME/src/3d/data/4dbar.dat
```

XtRXrt3dDouble

The resource converter calls the ANSI function *strtod()* to convert the resource specification to a double.

Contour Styles

Resources of type `XtRXrt3dContourStyles` are specified with any valid filename. The resource converter opens the file and calls *Xrt3dContourStylesFromFile()*. Any error messages are written to *stderr*.

Distribution Table

Resources of type `XtRXrt3dDistnTable` are specified with any valid filename. The resource converter opens the file and calls *Xrt3dDistnTableFromFile()*. Any error messages are written to *stderr*.

Grid Lines

Resources of type `XtRXrt3dGridLines` are specified as a combination of the strings `XYPlane`, `XZPlane`, and `YZPlane`, separated by plus signs. For example, to set X grid lines on both applicable planes:

```
*.XtXrt3d.xrt3dXGridLines: XYPlane + XZPlane
```

Images

Resources of type `XtRXrt3dImage` are specified with any valid filename. The filenames are checked for file type and are then converted to the proper image type if it's either a default image type or an image type registered by the user. An example:

```
*.XtXrt3d.xrt3dLegendImage: $XRTHOME/src/icons/shading.xpm
```

String Lists

Resources of type `XtRXrt3dStrings` are specified as a parenthesized list of strings, with each string contained in quotes. For example, to set the footer strings:

```
*.XtXrt3d.xrt3dFooterStrings: ("Two" "Lines!")
```

An empty list is specified by a single pair of parentheses.

Value-labels

Resources of type `XtRXrt3dValueLabels` 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:

```
*.XtXrt3d.xrt3dXValueLabels: (\
        (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. An empty list is designated by a single pair of parentheses.

XYColors

Resources of type `XtRXrt3dXYColors` are specified as a parenthesized list, with each member of the list specifying a complete XYColor within a set of parentheses. The color name must be quoted. An example:

```
*.XtXrt3d.xrt3dXYColors: (\
        (1 1 "red") \
        (2 -3 "green") \
        (6 8 "#FF23AC") \
    )
```

The entries are processed and stored in the order they appear in the list. Duplicate entries are removed in favor of later entries. An empty list is specified by a single pair of parentheses.

Projections

Resources of type `XtRXrt3dProject` are specified as a combination of the strings `ProjectContours` and `ProjectZones`, separated by plus signs. For example, to set contour and zone projection on the bottom of the unit cube:

```
*.XtXrt3d.xrt3dProjectZMin: ProjectContours + ProjectZones
```

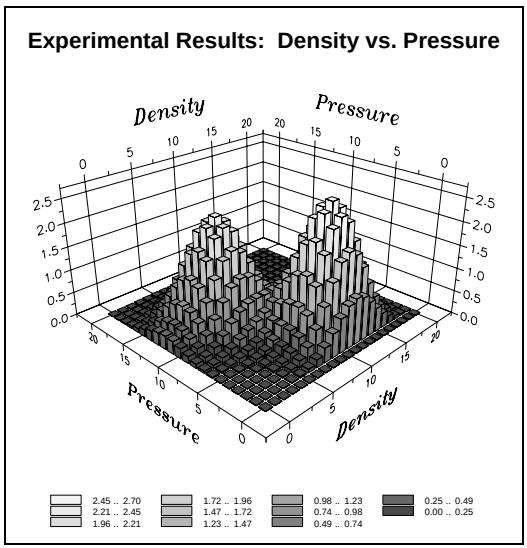
Text Objects

The list of XRT/3d text objects (`XtNxrt3dTextList`) is specified as a comma-separated list of text object names. The resource converter creates the text objects specified. Text object resources can be specified using the name of the text object, as shown by the following example:

```
*XtXrt3d.xrt3dTextList: text1,text2,text3
*XtXrt3dText.xrt3dTextBackgroundColor: blue
*XtXrt3dText.text2.xrt3dTextBackgroundColor: orange
```

G.2 Example Resource File

The following resource file defines the graph shown below:




```
! Sample resource file similar to $XRTHOME/src/3d/samples/4dbar.res
*xrt3dDrawMesh:             True
*xrt3dRepaint:              True
*xrt3dDrawShaded:           True
*xrt3dSolidSurface:         False
*xrt3dDrawZones:            True
*xrt3dSurfaceBottomColor:   dim grey
*xrt3dFooterAdjust:         AdjustCenter
*xrt3dSurfaceData:          $XRTHOME/src/3d/data/4dbar.dat
*xrt3dSurfaceTopColor:      light grey
*xrt3dFooterBorder:         BorderNone
*xrt3dFooterBorderWidth:    2
*xrt3dType:                 TypeBar
*xrt3dFooterFont:           XtDefaultFont
*xrt3dViewNormalized:       True
*xrt3dViewScale:            1
*xrt3dFooterHeight:         0
*height:                    582
*xrt3dViewXTranslate:       0
*xrt3dFooterStrings:        ()
*xrt3dViewYTranslate:       0
*xrt3dFooterWidth:          0
*xrt3dFooterX:              281
*xrt3dXAnnoMethod:          AnnoValues
*xrt3dFooterXUseDefault:    True
*xrt3dXAxisShow:            True
*xrt3dFooterY:              578
*xrt3dXAxisTitle:           Density
*xrt3dFooterYUseDefault:    True
*xrt3dXDataLabels:          ()
```

```
*xrt3dXGridLines:           xyplane+xzplane
*xrt3dForegroundColor:      black
*xrt3dGraphBackgroundColor: grey90
*xrt3dXMax:                 20
*xrt3dGraphBorder:          BorderNone
*xrt3dXMaxUseDefault:       False
*xrt3dXMeshFilter:          1
*xrt3dGraphBorderWidth:     2
*xrt3dGraphForegroundColor: black
*xrt3dXMeshShow:            True
*xrt3dGraphHeight:          436
*xrt3dXMin:                 0
*xrt3dGraphHeightUseDefault: True
*xrt3dXMinUseDefault:       False
*xrt3dXRotation:            25.0606
*xrt3dGraphWidth:           512
*xrt3dXScale:               1
*xrt3dGraphWidthUseDefault: True
*xrt3dXValueLabels:         ()
*xrt3dGraphX:               25
*xrt3dXYColors:             ()
*xrt3dGraphXUseDefault:     True
*xrt3dGraphY:               68
*xrt3dYAnnoMethod:          AnnoValues
*xrt3dGraphYUseDefault:     True
*xrt3dYAxisShow:            True
*xrt3dYAxisTitle:           Pressure
*xrt3dHeaderAdjust:         AdjustCenter
*xrt3dYDataLabels:          ()
*xrt3dYGridLines:           xyplane+yzplane
*xrt3dHeaderBorder:         BorderNone
*xrt3dHeaderBorderWidth:    2
*xrt3dYMax:                 20
*xrt3dYMaxUseDefault:       True
*xrt3dHeaderFont:           -*-helvetica-bold-r-normal--*-240-*-*-*-*-*-*
*xrt3dYMeshFilter:          1
*xrt3dYMeshShow:            True
*xrt3dHeaderHeight:         27
*xrt3dHeaderStrings:        ("Experimental Results:  Density vs. Pressure")
*xrt3dYMin:                 0
*xrt3dHeaderWidth:          512
*xrt3dYMinUseDefault:       True
*xrt3dYRotation:            -0.102286
*xrt3dHeaderX:              25
*xrt3dYScale:               1
*xrt3dHeaderXUseDefault:    True
*xrt3dYValueLabels:         ()
*xrt3dHeaderY:              22
*xrt3dHeaderYUseDefault:    True
*xrt3dZAnnoMethod:          AnnoValues
*xrt3dZAxisShow:            True
*xrt3dLegendAnchor:         AnchorSouth
*xrt3dZGridLines:           xzplane+yzplane
*xrt3dLegendBorder:         BorderNone
*xrt3dLegendBorderWidth:    2
*xrt3dZMax:                 2.69979
*xrt3dZMaxUseDefault:       True
*xrt3dLegendFont:           -*-helvetica-medium-r-normal--*-100-*-*-*-*-*-*
*xrt3dAxisStrokeFont:       SfRomanSimplex
*xrt3dLegendForegroundColor: black
```

```
*xrt3dZMin:                0
*xrt3dAxisStrokeSize:      55
*xrt3dLegendHeight:        48
*xrt3dZMinUseDefault:      True
*xrt3dAxisTitleStrokeFont: SfItalicTriplex
*xrt3dZOrigin:             0
*xrt3dAxisTitleStrokeSize: 95
*xrt3dZRotation:           45.3664
*xrt3dLegendOrientation:   AlignVertical
*xrt3dZScale:              0.5
*xrt3dLegendShow:          True
*xrt3dBackgroundColor:     white
*xrt3dZValueLabels:        ()
*xrt3dLegendStrings:       ()
*xrt3dXBarFormat:          BarFixed
*xrt3dLegendStyle:         LegendStyleStepped
*xrt3dZoneData:            $XRTHOME/src/3d/data/4dbar.dat
*xrt3dXBarSpacing:         80
*xrt3dZoneMethod:          ZoneContours
*xrt3dLegendWidth:         480
*xrt3dYBarFormat:          BarFixed
*width:                    562
*xrt3dLegendX:             39
*xrt3dYBarSpacing:         80
*xrt3dLegendXUseDefault:   True
*xrt3dLegendY:             523
*xrt3dDataAreaBackgroundColor: grey80
*xrt3dLegendYUseDefault:   True
*xrt3dDebug:               False
*xrt3dMeshBottomColor:     black
*xrt3dDistnMethod:         DistnLinear
*xrt3dMeshTopColor:        black
*xrt3dNumDistnLevels:      10
*xrt3dDoubleBuffer:        True
*xrt3dPerspectiveDepth:    2.5
*xrt3dDrawContours:        True
*xrt3dDrawHiddenLines:     False
```

H

UIL Arguments

This appendix provides an alphabetical listing of XRT/3d's UIL arguments and their data types. A second table provides XRT/3d UIL reasons. For more information on using XRT/3d with UIL, see section 8.10 on page 101.

UIL Argument Name	Argument Type
XtNxrt3dAxisStrokeFont	integer
XtNxrt3dAxisStrokeSize	integer
XtNxrt3dAxisTitleStrokeFont	integer
XtNxrt3dAxisTitleStrokeSize	integer
XtNxrt3dBackgroundColor	string
XtNxrt3dDataAreaBackgroundColor	string
XtNxrt3dDataStylesUseDefault	boolean
XtNxrt3dDebug	boolean
XtNxrt3dDistnMethod	integer
XtNxrt3dDoubleBuffer	boolean
XtNxrt3dDrawContours	boolean
XtNxrt3dDrawDropLines	boolean
XtNxrt3dDrawHiddenLines	boolean
XtNxrt3dDrawMesh	boolean
XtNxrt3dDrawShaded	boolean
XtNxrt3dDrawZones	boolean
XtNxrt3dFooterAdjust	integer
XtNxrt3dFooterBackgroundColor	string

UIL Argument Name	Argument Type
XtNxrt3dFooterBorder	integer
XtNxrt3dFooterBorderWidth	integer
XtNxrt3dFooterFont	font_table
XtNxrt3dFooterForegroundColor	string
XtNxrt3dFooterHeight	integer
XtNxrt3dFooterImage	pixmap
XtNxrt3dFooterImageLayout	integer
XtNxrt3dFooterImageMinimumSize	boolean
XtNxrt3dFooterWidth	integer
XtNxrt3dFooterX	integer
XtNxrt3dFooterXUseDefault	boolean
XtNxrt3dFooterY	integer
XtNxrt3dFooterYUseDefault	boolean
XtNxrt3dForegroundColor	string
XtNxrt3dGraphBackgroundColor	string
XtNxrt3dGraphBorder	integer
XtNxrt3dGraphBorderWidth	integer
XtNxrt3dGraphForegroundColor	string
XtNxrt3dGraphHeight	integer
XtNxrt3dGraphHeightUseDefault	boolean
XtNxrt3dGraphImage	pixmap
XtNxrt3dGraphImageLayout	integer
XtNxrt3dGraphWidth	integer
XtNxrt3dGraphWidthUseDefault	boolean
XtNxrt3dGraphX	integer
XtNxrt3dGraphXUseDefault	boolean
XtNxrt3dGraphY	integer
XtNxrt3dGraphYUseDefault	boolean
XtNxrt3dHeaderAdjust	integer
XtNxrt3dHeaderBackgroundColor	string
XtNxrt3dHeaderBorder	integer
XtNxrt3dHeaderBorderWidth	integer

UIL Argument Name	Argument Type
XtNxrt3dHeaderFont	font_table
XtNxrt3dHeaderForegroundColor	string
XtNxrt3dHeaderHeight	integer
XtNxrt3dHeaderImage	pixmap
XtNxrt3dHeaderImageLayout	integer
XtNxrt3dHeaderImageMinimumSize	boolean
XtNxrt3dHeaderWidth	integer
XtNxrt3dHeaderX	integer
XtNxrt3dHeaderXUseDefault	boolean
XtNxrt3dHeaderY	integer
XtNxrt3dHeaderYUseDefault	boolean
XtNxrt3dImage	pixmap
XtNxrt3dImageLayout	integer
XtNxrt3dLegendAnchor	integer
XtNxrt3dLegendBackgroundColor	string
XtNxrt3dLegendBorder	integer
XtNxrt3dLegendBorderWidth	integer
XtNxrt3dLegendFont	font_table
XtNxrt3dLegendForegroundColor	string
XtNxrt3dLegendHeight	integer
XtNxrt3dLegendImage	pixmap
XtNxrt3dLegendImageLayout	integer
XtNxrt3dLegendLabelFilter	integer
XtNxrt3dLegendOrientation	integer
XtNxrt3dLegendShow	boolean
XtNxrt3dLegendStyle	integer
XtNxrt3dLegendTitle	string
XtNxrt3dLegendWidth	integer
XtNxrt3dLegendX	integer
XtNxrt3dLegendXUseDefault	boolean
XtNxrt3dLegendY	integer
XtNxrt3dLegendYUseDefault	boolean

UIL Argument Name	Argument Type
XtNxrt3dMeshBottomColor	string
XtNxrt3dMeshTopColor	string
XtNxrt3dNumDistnLevels	integer
XtNxrt3dPerspectiveDepth	float
XtNxrt3dPreviewMethod	integer
XtNxrt3dProjectZMax	integer
XtNxrt3dProjectZMin	integer
XtNxrt3dRepaint	boolean
XtNxrt3dSolidSurface	boolean
XtNxrt3dSurfaceBottomColor	string
XtNxrt3dSurfaceTopColor	string
XtNxrt3dTextAdjust	integer
XtNxrt3dTextAttachIndexPoint	integer
XtNxrt3dTextAttachIndexSeries	integer
XtNxrt3dTextAttachIndexX	integer
XtNxrt3dTextAttachIndexY	integer
XtNxrt3dTextAttachMethod	integer
XtNxrt3dTextAttachPixelX	integer
XtNxrt3dTextAttachPixelY	integer
XtNxrt3dTextAttachPointX	float
XtNxrt3dTextAttachPointY	float
XtNxrt3dTextAttachPointZ	float
XtNxrt3dTextBackgroundColor	string
XtNxrt3dTextBorder	integer
XtNxrt3dTextBorderWidth	integer
XtNxrt3dTextFont	font_table
XtNxrt3dTextForegroundColor	string
XtNxrt3dTextImage	pixmap
XtNxrt3dTextImageLayout	integer
XtNxrt3dTextImageMinimumSize	boolean
XtNxrt3dTextLineShow	boolean
XtNxrt3dTextOffsetX	integer

UIL Argument Name	Argument Type
XtNxrt3dTextOffsetY	integer
XtNxrt3dTextPlane	integer
XtNxrt3dTextPrintFont	string
XtNxrt3dTextShow	boolean
XtNxrt3dTextStrokeFont	integer
XtNxrt3dTextStrokeSize	integer
XtNxrt3dType	integer
XtNxrt3dViewNormalized	boolean
XtNxrt3dViewScale	float
XtNxrt3dViewXTranslate	float
XtNxrt3dViewYTranslate	float
XtNxrt3dXAnnoMethod	integer
XtNxrt3dXAxisShow	boolean
XtNxrt3dXAxisTitle	string
XtNxrt3dXBarFormat	integer
XtNxrt3dXBarSpacing	float
XtNxrt3dXGridLines	integer
XtNxrt3dXGridLineStyleUseDefault	boolean
XtNxrt3dXMax	float
XtNxrt3dXMaxUseDefault	boolean
XtNxrt3dXMeshFilter	integer
XtNxrt3dXMeshShow	boolean
XtNxrt3dXMin	float
XtNxrt3dXMinUseDefault	boolean
XtNxrt3dXRotation	float
XtNxrt3dXScale	float
XtNxrt3dYAnnoMethod	integer
XtNxrt3dYAxisShow	boolean
XtNxrt3dYAxisTitle	string
XtNxrt3dYBarFormat	integer
XtNxrt3dYBarSpacing	float
XtNxrt3dYGridLines	integer

UIL Argument Name	Argument Type
XtNxrt3dYGridLineStyleUseDefault	boolean
XtNxrt3dYMax	float
XtNxrt3dYMaxUseDefault	boolean
XtNxrt3dYMaxUseDefault	integer
XtNxrt3dYMeshShow	boolean
XtNxrt3dYMin	float
XtNxrt3dYMinUseDefault	boolean
XtNxrt3dYRotation	float
XtNxrt3dYScale	float
XtNxrt3dZAnnoMethod	integer
XtNxrt3dZAxisShow	boolean
XtNxrt3dZAxisTitle	string
XtNxrt3dZGridLines	integer
XtNxrt3dZGridLineStyleUseDefault	boolean
XtNxrt3dZMax	float
XtNxrt3dZMaxUseDefault	boolean
XtNxrt3dZMin	float
XtNxrt3dZMinUseDefault	boolean
XtNxrt3dZoneMethod	integer
XtNxrt3dZOrigin	float
XtNxrt3dZRotation	float
XtNxrt3dZScale	float

Reasons
XtNxrt3dExposeCallback
XtNxrt3dMapCallback
XtNxrt3dModifyCallback
XtNxrt3dPickCallback
XtNxrt3dResizeCallback
XtNxrt3dRotateCallback
XtNxrt3dTransformCallback

Index

15 graph types 43, 57
2-byte fonts, using 98
4D graphs 87, 88
  creating 87

A

action routines
  DrawGraph() 74, 188
  DrawPreview() 74, 188
  Map() 79, 188
  ModifyCancel() 188
  ModifyEnd() 78, 189
  ModifyStart() 73, 188
  NoAction() 77, 189
  Pick() 78, 189
  RotateUpdate() 76, 189
  ScaleUpdate() 74, 190
  SelectAxis() 190
  TranslateUpdate() 74, 190
  ViewReset() 76, 190
  ZoomCancel() 191
  ZoomEnd() 74, 191
  ZoomUpdate() 190
actions, listing of 187
ANSI C, compatibility with XRT/3d 11
arguments, UIL 223
authorizing
  shared libraries 11
authorizing XRT/3d programs 10
axis
  displaying 29
  font and size 29
  labelling 32
  minimum/maximum bounds 29
  scaling 31
  title 29
axis labels
  data labels method 32
  value labels method 32
  value method 32
axis resources, summary 25

B

background color
  Text Object 92
background colors 36
  data area 36
  footer 36
  graph 36
  header 36
  legend 36
  window 36
background image
  minimum image size 40
bar charts
  4D 88
  coloring 50
bars.c example 13
batching resource updates 19
border types 36
borders 36
  Text Object 92

C

C++
  compatibility with XRT/3d 11
  header file 101
  use of 101
C++ classes
  Xrt3d 101
  Xrt3dDataC 101
  Xrt3dText 101
callback resources, summary 27
callback structures
  Expose 138
  Map 146, 197
  Modify 146, 198
  Pick 147, 198
  Resize 148
  Rotate 149, 199
  Transform 150, 151, 203
callbacks
  Modify 189
  reasons 199
  Transform 190, 191
CGM output 41
changes in this release 2
chart
  background color 36
class hierarchy 133
class name 133

class pointer 133
  XRT/3d Text Object 157
color field, in XrtDataStyle 59
Colors
  Get Closest Color 37
colors
  background 36
  bar charts 50
  color database 20
  flashing of colormap 20
  foreground 37
  graph mesh 54
  graph surface 55
  monochrome use 20
  specifying 20
  strategies for use of 38
  Text Object 92
comments on product 4
compiling and authorizing 10
compiling XRT/3d programs 10
contour graphs 45
contour styles
  customizing 95, 96
  line patterns 96
  usage tips 97
contours 44
contours and zones graphs 47
controlling interactive rotate 76
controlling interactive scale, transform, zoom 75
cross-platform applications with Microsoft Windows 98

D

data
  application responsibility 65
  changing 65
  convenience procedures 69
  example of setting a value 68
  input from file 64
  irregular grid 67
  overview of Xrt3dData structure 63
  real-time 65
  real-time performance 65
  regular grid 66
  scatter plot 68
data area
  background color 36
data display resources, summary 24
Data styles 58
  accessing 61
  default 58
  line patterns 59
  setting 60
  setting for monochrome display 60
  symbol styles 60
debugging XRT/3d applications 21
default user interactions 71
demo programs, discussion of 14
destroy procedure 112
Dimension 35
disabling all user interaction 74
disabling types of user interaction 77
distribution table, customizing 93
double buffering 40
DrawContours resource 44
DrawGraph() action 74, 188
DrawMesh resource 43
DrawPreview() action 74, 188
DrawShaded resource 44
DrawZones resource 44
DrawZones with DrawShaded 48
dynamic and static colors, discussion 20

E

environment variables, XRTHOME 9
event handling procedures 80
example programs, listing of 13
Expose callback 84

F

feedback3d demo 14
floating point properties, specifying 17
floating point resources, specifying 17
fonts
  2-byte 98
  header, footer, legend 34
  Text Object 92
footer
  background color 36
  border 36
  font 34
  foreground color 37
  image 38
  positioning 35
  text 33
footer area, definition of 15
foreground color
  Text Object 92
foreground colors 37
  footer 37
  graph 37
  header 37
  legend 37
  window 37

G

graph
  background color 36
  border 36
  foreground color 37
  grid lines 33
  image 38
  output 40

perspective effect 31
positioning 35
positioning of areas 35
graph area, definition of 15
graph areas
illustration of 17
positioning resource summary 35
graph data area
background color 36
graph mesh
colors 54
filtering 54
hidden lines 55
graph resources, summary 25
graph surface
colors 55
solid 55
graph types 43, 57
4D 87, 88
contours 45
contours and zones 47
DrawContours 44
DrawMesh 43
DrawShaded 44
DrawZones 44
mesh 45
mesh and contour 46
mesh and shaded 46
mesh and zones 46
mesh, contours, zones 47
mesh, shaded, contours 47
mesh, shaded, contours, zones 48
mesh, shaded, zones 47
shaded 45
shaded and contour 46
shaded and zones 46
shaded, contours, zones 47
zones 45
grid lines 33
Grid text objects 89
characteristics 90
GUI builders
resource setting syntax 217

H

header
background color 36
border 36
font 34
foreground color 37
image 38
positioning 35
text 33
header area, definition of 15
header file, XRT/3d 133
header/footer resources, summary 26
holes in data 64

I

image 38
include file, XRT/3d 133
instantiating graph widget 133
internationalization
2-byte fonts 98
ISO Latin 1 character set 98
introduction to XRT/3d 1
irregular grid data 67
ISO Latin 1 character set 98

L

legend
4D graphs 89
anchor 30
background color 36
bar chart colors 50
border 30, 36
customizing contents 31
explicit positioning 30
font 34
foreground color 37
image 38
layout 30
orientation 30
positioning 30, 35
style 30
legend area, definition of 15
legend resources, summary 27
line patterns 59, 96
linking
shared libraries 11

M

Map callback 80
Map callback structure 197
Map() action 79, 145, 188
use of Xrt3dMap() 79
math3d demo 14
mesh
colors 54
filtering 54
hidden lines 55
mesh and contour graphs 46
mesh and shaded graphs 46
mesh and zones graphs 46
mesh graphs 45
mesh, contours, zones graphs 47
mesh, shaded, contours graphs 47
mesh, shaded, contours, zones graphs 48
mesh, shaded, zones graphs 47
miscellaneous resources, summary 28
Modify callback 74, 78, 189
Modify callback structure 198
ModifyCancel() action 188

ModifyEnd() action 78, 146, 189
ModifyStart() action 73, 146, 188
monochrome display
  colors 20
  Data styles setting caution 60
Motif class hierarchy 16
Motif integration 15
mvc demo 14

N

nentries, Xrt3dDistnTable structure 93
nlevels 93, 95, 97
NoAction() action 77, 189
noval element of Xrt3dData structure 64

O

outputting graphs 40
  sample code 41

P

pattern field, in XrtDataStyle 59
perspective effect 31
Pick callback 79
Pick callback structure 198
Pick() action 78, 147, 189
  use of Xrt3dPick() 78
Pixel text objects 90
  characteristics 90
play3d demo 14
pointer resources, specifying 18
porting to Microsoft Windows 98
Position 35
positioning graph areas, strategies 35
PostScript output 40
printing graphs 40
  sample code 41
product feedback 4
profile demo 14
properties
  floating point 17
property files
  example 220
  syntax 217

R

real-time data 65
real-time performance 65
regular grid data 66
resetting graph
  default translation 187
resetting interactions 76
Resize callback 84
resizing windows 84
resource files
  comments in 217
  continuing lines 217
  discussion of use 21
  doubles 218
  example 220
  specifying Booleans 217
  specifying contour styles 218
  specifying distribution table 218
  specifying enumerated values 217
  specifying graph data 218
  specifying grid lines 218
  specifying projections 219
  specifying string lists 218
  specifying text object resources 219
  specifying Value-labels 219
  specifying XYColors 219
  syntax 217
  use of 217
resource summary
  axis 25
  callback 27
  data display 24
  graph 25
  header/footer 26
  legend 27
  miscellaneous 28
  Text Object 28
resource type/class listing 205
resources
  batching updates 19
  floating point 17
  pointer 18
  UseDefault 18
Rotate callback 76
Rotate callback structure 199
RotateUpdate() action 76, 189
rotating, interactive 71
rotation
  default translation 187

S

sample code 13
scale of axis 31
ScaleUpdate() action 74, 190
scaling
  default translation 187
scaling, interactive 71
scatter plot data 68
SelectAxis() action 190
shaded and contour graphs 46
shaded and zones graphs 46
shaded graphs 45
shaded, contours, zones graphs 47
shader demo 14
shared libraries
  linking and authorizing 11
simple example 9

simple.c example 13
  discussion of 11
simple_cpp.cxx example 13
simple_uil.c example 13
size field, in XrtDataStyle 59
spline demo 14
static and dynamic colors, discussion 20
stroke fonts, listing of 200
surface
  colors 55
  solid 55
surface graphs
  4D 87
  creating 4D 87
surface_editor demo 14
symbol styles 60

T

technical support 4
terminology of XRT/3d 17
Text Object
  attachment methods 89
  characteristics of Grid types 90
  characteristics of Pixel type 90
  class hierarchy 157
  class name 157
  class pointer 157
  definition 89
  discussion of Xt Object class 89
  functions/macros 157
  instantiating 157
  intersecting multiple objects 90
  positioning 91
  removing 93
  resource summary 90
  synopsis 157
  types of 89
Text Object resources, summary 28
texts.c example 13
Transform callback 75, 190, 191
  example of use 75
Transform callback structure 203
TranslateUpdate() action 74, 149, 150, 190
translation
  default translation 187
translation, interactive 71
translations and actions 73
  disabling 74
  programming overview 73
  rotating 71
  scaling 71
  translation 71
  zoom 71
  see also user interaction
translations, listing of 187
types of graphs 43, 57

U

UIL
  example 102
  header file 102
  reasons 228
  use of 101
UIL arguments 223
UseDefault resources 18
user interaction
  controlling rotate 76
  controlling scale, translate, zoom 75
  customizing 73
  default translations and actions 187
  disabling all 74
  disabling particular interactions 77
  disabling types of interaction 77
  ending 78
  event handling 80
  resetting 76
  rotating 71
  scaling 71
  starting 73
  three stages 73
  translation 71
  updating 74
  with graph data 78
  zooming 71
user interaction, default 71
using C++ with XRT/3d 101
using GUI Builders 217
using resource files 217
using UIL with XRT/3d 101

V

view_dir demo 14
ViewReset() action 76, 149, 150, 190

W

widget
  class hierarchy 133
  class name 133
  class pointer 133
  debugging 21
  functions/macros 133
  include file 133
  instantiating 133
  synopsis 133
width field, in XrtDataStyle 59
windows, resizing 84

X

XAllocColor() 37
XFreeColors() 37
XImage representation of graph 41
XListFonts() 34
XmNxrtPDSSorted 124
XmPrimitive 15
XRT/3d
  basic terminology 17
  changes in this release 2
  class hierarchy 133
  class name 133
  class pointer 133
  functions/macros 133
  include file 133
  instantiating 133
  introduction 9
  introduction to 1
  resource types/classes 205
XRT/3d Text Object
  attachment methods 89
  characteristics of Grid types 90
  characteristics of Pixel type 90
  class hierarchy 157
  class name 157
  class pointer 157
  definition of 89
  discussion of Xt Object class 89
  functions/macros 157
  instantiating 157
  intersecting multiple objects 90
  positioning 91
  removing 93
  resource summary 90
  synopsis 157
  types 89
XRT/3d Text Object resources, summary 28
xrt_auth program
  flags 10
  use of 10
XRT3D_AXIS_TITLE_STROKE_FONT 134
XRT3D_GRAPH_WIDTH_USE_DEFAULT 141
XRT3D_HEADER_FOREGROUND_COLOR 37
XRT3D_PREVIEW_METHOD 147
XRT3D_XSCALE 155
Xrt3dAdjust 193
Xrt3dAlign 193
Xrt3dAnchor 193
Xrt3dAnnoMethod 32, 193
Xrt3dArgValToDouble() 161
  use when porting to Windows 99
Xrt3dAttachWindow() 99
Xrt3dAxis 194
Xrt3dBarFormat 194
Xrt3dBorder 194
Xrt3dCallbackStruct structure 138, 148, 194
Xrt3dComputeZValue 83
Xrt3dComputeZValue() 161
Xrt3dContourStyle structure 95, 162, 168, 171, 180, 195
  fill_color 96
  line_color 96
  line_width 96
  lpat 96
Xrt3dContourStylesFromFile() 162
Xrt3dCreate() 99
Xrt3dData structure 63, 164, 174, 176, 195
  application responsibility 65
  changing data 65
  convenience procedures 69
  description of 66
  example of setting 68
  input from file 64
  irregular grid 67
  overview 63
  regular grid 66
  scatter plot 68
  use of as 4D surface 87
Xrt3dDataCopy() 162
Xrt3dDataShaded() 162
Xrt3dDataSmooth() 163
Xrt3dDataStyle structure 168, 195
Xrt3dDataType 195
Xrt3dDataUpdateDataValue() 163
Xrt3dDataWindow() 163
Xrt3dDestroyData() 164
Xrt3dDistnIndex() 164
Xrt3dDistnMethod 195
Xrt3dDistnRange 195
Xrt3dDistnTable structure 93, 136, 164, 168, 196
Xrt3dDistnTableFromFile() 164
Xrt3dDoubleToArgVal() 17, 165
  use when porting to Windows 99
Xrt3dDrawCGM() 41, 165
  XtNxrt3dTextPrintFont resource 160
Xrt3dDrawPS() 40, 166
  internationalization support 98
  XtNxrt3dTextPrintFont resource 160
Xrt3dDrawPSColor 196
Xrt3dDupContourStyles() 18, 96, 168
Xrt3dDupDataStyles() 168
Xrt3dDupDistnTable() 18, 168
Xrt3dDupStrings() 18, 168
Xrt3dDupValueLabels() 18, 169
Xrt3dDupXYColors() 18, 169
Xrt3dFooterImage 139
Xrt3dFooterImageLayout 139
Xrt3dFreeContourStyles() 169
Xrt3dFreeDataStyles() 169
Xrt3dFreeDistnTable() 170
Xrt3dFreePropString() 170
Xrt3dFreeStrings() 170
Xrt3dFreeValueLabels() 170
Xrt3dFreeXYColors() 170
Xrt3dGetDistnTable() 170
Xrt3dGetImage() 41, 171
Xrt3dGetNthContourStyle() 97, 171
Xrt3dGetNthDataLabel() 171

Xrt3dGetNthDataStyle() 171
Xrt3dGetNthFooterString() 172
Xrt3dGetNthHeaderString() 172
Xrt3dGetNthLegendString() 172
Xrt3dGetPropString() 172, 199
Xrt3dGetValueLabel() 33, 173
Xrt3dGetValues() 99, 173, 199
Xrt3dGetXYColor() 50, 173
Xrt3dGraphImage 140
Xrt3dGraphImageLayout 140
Xrt3dGridData structure 66, 196
Xrt3dHeaderImage 142
Xrt3dHeaderImageLayout 142
Xrt3dImage 142
Xrt3dImage_Layout 143
Xrt3dImageLayout 196
Xrt3dInsertNthDataStyle() 174
Xrt3dInterpMethod 196
Xrt3dIrGridData structure 67, 197
Xrt3dLegendImage 143
Xrt3dLegendImageLayout 144
Xrt3dLegendStyle 197
Xrt3dLinePattern 95, 197
Xrt3dLineStyle 197
Xrt3dMakeDataFromFile() 66, 67, 68, 174
Xrt3dMakeGridData() 66, 67, 176
Xrt3dMakeIrGridData() 176
Xrt3dMakePointData() 68, 176
Xrt3dMakePointDataSeries() 177
Xrt3dMap() 82, 177
  use of by Map() action 188
Xrt3dMapCallbackStruct structure 80, 146, 197
Xrt3dMapResult structure 83, 198
Xrt3dModifyCallbackStruct structure 74, 78, 146, 198
Xrt3dMrmInitialize() 102, 177
Xrt3dOutputPS() 41, 177
Xrt3dOutputXwd() 41, 178
Xrt3dPick() 80, 178, 189
Xrt3dPickCallbackStruct structure 79, 147, 198
Xrt3dPickResult structure 81, 198
Xrt3dPoint3D 198
Xrt3dPointData 198
Xrt3dPointData structure 68
Xrt3dPointSeries 199
Xrt3dPreviewMethod 199
Xrt3dProperty 199
  list of enumerations 205
Xrt3dReason 199
Xrt3dRegion 199
Xrt3dRemoveNthDataStyle() 178
Xrt3dResetContourStyles() 179
Xrt3dRotateCallbackStruct structure 76, 149, 199
Xrt3dSaveDataToFile() 179
Xrt3dSetNthContourStyle() 180
Xrt3dSetNthDataLabel() 181
Xrt3dSetNthDataStyle() 181
Xrt3dSetNthFooterString() 181
Xrt3dSetNthHeaderString() 182
Xrt3dSetNthLegendString() 182
Xrt3dSetPropString() 182, 199
  resource syntax 217
Xrt3dSetReverseConverters() 102, 182
Xrt3dSetValueLabel() 33, 183
Xrt3dSetValues() 99, 183, 199
Xrt3dSetXYColor 50
Xrt3dSetXYColor() 183
Xrt3dStrokeFont 200
Xrt3dSymbolPattern 202
Xrt3dSymbolStyle 202
Xrt3dTextAttachMethod 203
Xrt3dTextGetPropString() 184
Xrt3dTextGetValues() 184
Xrt3dTextImage 159
Xrt3dTextImageLayout 159
Xrt3dTextSetPropString() 184
Xrt3dTextSetValues() 185
Xrt3dTransformCallbackStruct structure 75, 150, 151, 203
Xrt3dType 203
Xrt3dUnmap() 83, 185
Xrt3dUnpick() 82, 185
Xrt3dValueLabel structure 33, 203
Xrt3dXYColor structure 50, 203
Xrt3dZInterpolate() 186
Xrt3dZoneMethod 203
XrtDataStyle structure 59
  color field 59
  pattern field 59
  size field 59
  width field 59
XrtDupDataStyles() 61
XrtGetNthDataStyle() 61
XrtGetPropString() 100
XRTHOME environment variable 9
XrtList
  adding items 111
  adding sorted items 112
  checking for existence 113
  comparing items 115
  copying 111
  creating 111
  destroy procedure 112
  destroying 111
  destroying item memory 112
  introduction 110
  item count 115
  moving items 117
  removing items 112
  retrieving entire array 115
  retrieving item by index 114
  sample code 110
  searching 113
  sorting 116
XrtPDSCmpDoubles() 116, 120
XrtPDSCmpFloats() 116, 120
XrtPDSCmpInts() 116, 121
XrtPDSCmpStrings() 116, 121
XrtPDSCmpStringsIgnoreCase() 116, 121
XrtPDSListAdd() 112, 121

XrtPDSListAppend() 111, 122
XrtPDSListCreate() 111, 122
XrtPDSListDelete() 112, 122, 127
XrtPDSListDeleteAll() 122, 127
XrtPDSListDestroy() 111, 112, 123
XrtPDSListDestroyAll() 112
XrtPDSListDestroyObject() 112, 123
XrtPDSListDestroyXtPointer() 113, 123
XrtPDSListDuplicate() 111, 123
XrtPDSListFind() 113, 115, 123
XrtPDSListGetCompareProc() 114, 121, 123, 124, 126, 127, 128
XrtPDSListGetItem() 114, 124
XrtPDSListGetItemCount() 115, 124
XrtPDSListGetItems() 115, 124
XrtPDSListGetItemSize() 115, 121, 122, 124, 125
XrtPDSListGetUserData() 117, 125
XrtPDSListInsertAfter() 111, 125
XrtPDSListInsertBefore(111
XrtPDSListInsertBefore() 111, 125
XrtPDSListIsList() 113, 125
XrtPDSListIsSorted() 117, 126
XrtPDSListLookupItem() 113, 126
XrtPDSListMoveItemsAfter() 126
XrtPDSListMoveItemsBefore() 117, 126
XrtPDSListRemove() 112, 122, 127
XrtPDSListRemoveAll() 112, 122, 127
XrtPDSListSearchBackward() 115
XrtPDSListSearchForward() 115, 127
XrtPDSListSearchReverse() 128
XrtPDSListSetCompareProc() 113, 115, 116, 120, 121, 128
XrtPDSListSetDestroyProc() 112, 123, 128
XrtPDSListSetItem() 114, 129
XrtPDSListSetUserData() 117, 129
XrtPDSListSort() 116, 129
XrtPDSSetDestroyProc() 122, 127
XrtSampleColorFunction() 37
XrtSetNthDataStyle() 61
XrtSetPropString() 100
XtAddCallback()
  example of use 75
XtConvertAndStore() 34
  use with XRT/3d reverse converters 102, 182
XtCreateXrt3d() 186
XtCreateXrt3dText() 186
XtIsXrt3d() 186
XtIsXrt3dText() 186
XtMxrt3dTransformCallback 190
XtNxrt3dAxisStrokeFont 29, 134
XtNxrt3dAxisStrokeSize 134
XtNxrt3dAxisTitleStrokeFont 134
XtNxrt3dAxisTitleStrokeSize 134
XtNxrt3dBackgroundColor 134
XtNxrt3dBorder 134
XtNxrt3dContourStyles 44, 134
XtNxrt3dDataAreaBackgroundColor 135
XtNxrt3dDataStyles 135
XtNxrt3dDataStylesUseDefault 135
XtNxrt3dDebug 21, 135
XtNxrt3dDistnMethod 44, 93, 135
XtNxrt3dDistnTable 44, 136, 164
XtNxrt3dDoubleBuffer 40, 136
XtNxrt3dDrawContours 136
XtNxrt3dDrawDropLines 136
XtNxrt3dDrawHiddenLines 44, 55, 136
XtNxrt3dDrawMesh 137
XtNxrt3dDrawShaded 137
XtNxrt3dDrawZones 137
XtNxrt3dExposeCallback 84, 137
XtNxrt3dFooterAdjust 33, 138
XtNxrt3dFooterBackgroundcolor 138
XtNxrt3dFooterBorder 138
XtNxrt3dFooterBorderWidth 139
XtNxrt3dFooterFont 139
XtNxrt3dFooterForegroundColor 139
XtNxrt3dFooterHeight 139
XtNxrt3dFooterStrings 33, 139
XtNxrt3dFooterWidth 139
XtNxrt3dFooterX 139
XtNxrt3dFooterXUseDefault 139
XtNxrt3dFooterY 139
XtNxrt3dFooterYUseDefault 139
XtNxrt3dForegroundColor 140
XtNxrt3dGraphBackgroundColor 140
XtNxrt3dGraphBorder 140
XtNxrt3dGraphBorderWidth 140
XtNxrt3dGraphForegroundColor 140
XtNxrt3dGraphHeight 140
XtNxrt3dGraphHeightUseDefault 141
XtNxrt3dGraphWidth 140
XtNxrt3dGraphX 141
XtNxrt3dGraphXUseDefault 141
XtNxrt3dGraphY 141
XtNxrt3dGraphYUseDefault 141
XtNxrt3dHeaderAdjust 33, 141
XtNxrt3dHeaderBackgroundColor 141
XtNxrt3dHeaderBorder 141
XtNxrt3dHeaderBorderWidth 141
XtNxrt3dHeaderFont 141
XtNxrt3dHeaderForegroundColor 142
XtNxrt3dHeaderHeight 142
XtNxrt3dHeaderImageMinimumSize 139, 142
XtNxrt3dHeaderStrings 33, 142
XtNxrt3dHeaderWidth 142
XtNxrt3dHeaderX 142
XtNxrt3dHeaderXUseDefault 142
XtNxrt3dHeaderY 142
XtNxrt3dHeaderYUseDefault 142
XtNxrt3dLegendAnchor 35, 143
XtNxrt3dLegendBackgroundColor 143
XtNxrt3dLegendBorder 143
XtNxrt3dLegendBorderWidth 143
XtNxrt3dLegendDistnRange 143
XtNxrt3dLegendFont 143
XtNxrt3dLegendForegroundColor 143
XtNxrt3dLegendHeight 145
XtNxrt3dLegendLabelFilter 144
XtNxrt3dLegendLabelFunc 95, 144
XtNxrt3dLegendLabelTitle 145

XtNxrt3dLegendOrientation 30, 144
XtNxrt3dLegendShow 29, 144
XtNxrt3dLegendStrings 89, 94, 144
XtNxrt3dLegendStyle 30, 145
XtNxrt3dLegendWidth 145
XtNxrt3dLegendX 145
XtNxrt3dLegendXUseDefault 145
XtNxrt3dLegendY 145
XtNxrt3dLegendYUseDefault 145
XtNxrt3dMapCallback 80, 145
XtNxrt3dMeshBottomColor 44, 54, 146
XtNxrt3dMeshTopColor 44, 54, 146
XtNxrt3dModifyCallback 78, 146, 189
XtNxrt3dNumDistnLevels 93, 147
XtNxrt3dPerspectiveDepth 31, 147
XtNxrt3dPickCallback 79, 147
XtNxrt3dProjectZMax 52, 148
XtNxrt3dProjectZMin 52, 148
XtNxrt3dRepaint 19, 148
XtNxrt3dResizeCallback 84, 148
XtNxrt3dRotateCallback 148
XtNxrt3dSolidSurface 55, 149
XtNxrt3dSurfaceBottomColor 55, 149
XtNxrt3dSurfaceData 18, 65, 66, 149
XtNxrt3dSurfaceTopColor 55, 149
XtNxrt3dTextAdjust 91, 157
XtNxrt3dTextAttachIndexPoint 158
XtNxrt3dTextAttachIndexSeries 158
XtNxrt3dTextAttachIndexX 158
XtNxrt3dTextAttachIndexY 158
XtNxrt3dTextAttachMethod 89, 158
XtNxrt3dTextAttachPixelX 158
XtNxrt3dTextAttachPixelY 158
XtNxrt3dTextAttachPointX 158
XtNxrt3dTextAttachPointY 158
XtNxrt3dTextAttachPointZ 158
XtNxrt3dTextBackground 92
XtNxrt3dTextBackgroundColor 158
XtNxrt3dTextBorder 92, 158
XtNxrt3dTextBorderWidth 92, 159
XtNxrt3dTextFont 92, 159
XtNxrt3dTextForeground 92
XtNxrt3dTextForegroundColor 159
XtNxrt3dTextImageMinimumSize 159
XtNxrt3dTextLineShow 92, 159
XtNxrt3dTextList 149, 150
XtNxrt3dTextOffsetX 159
XtNxrt3dTextOffsetY 159
XtNxrt3dTextPlane 91, 160
XtNxrt3dTextPrintFont 160
XtNxrt3dTextShow 93, 160
XtNxrt3dTextStrings 91, 160
XtNxrt3dTextStrokeFont 92, 160
XtNxrt3dTextStrokeSize 92, 160
XtNxrt3dTransformCallback 75, 149, 150, 190, 191
XtNxrt3dType 48, 150
XtNxrt3dViewNormalized 151
XtNxrt3dViewScale 151
XtNxrt3dViewXTranslate 151
XtNxrt3dViewYTranslate 151
XtNxrt3dXAnnoMethod 32, 152
XtNxrt3dXAxisShow 29, 152
XtNxrt3dXAxisTitle 152
XtNxrt3dXBarFormat 152
XtNxrt3dXBarSpacing 152
XtNxrt3dXDataLabels 152
XtNxrt3dXGridLines 33, 153
XtNxrt3dXGridLineStyle 153
XtNxrt3dXGridLineStyleUseDefault 153
XtNxrt3dXMax 29, 153
XtNxrt3dXMaxUseDefault 29, 153
XtNxrt3dXMeshFilter 54, 154
XtNxrt3dXMeshShow 154
XtNxrt3dXMin 29, 154
XtNxrt3dXMinUseDefault 29, 154
XtNxrt3dXRotation 76, 154
XtNxrt3dXScale 31, 155
XtNxrt3dXTitle 29
XtNxrt3dXValueLabels 155
XtNxrt3dXYColors 50, 155
XtNxrt3dYAnnoMethod 32, 152
XtNxrt3dYAxisShow 29, 152
XtNxrt3dYAxisTitle 152
XtNxrt3dYBarFormat 152
XtNxrt3dYBarSpacing 152
XtNxrt3dYDataLabels 152
XtNxrt3dYGridLines 33, 153
XtNxrt3dYMax 29, 153
XtNxrt3dYMaxUseDefault 29, 153
XtNxrt3dYMeshFilter 54, 154
XtNxrt3dYMeshShow 154
XtNxrt3dYMin 29, 154
XtNxrt3dYMinUseDefault 29, 154
XtNxrt3dYRotation 76, 154
XtNxrt3dYScale 31, 155
XtNxrt3dYTitle 29
XtNxrt3dYValueLabels 155
XtNxrt3dZAnnoMethod 32, 152
XtNxrt3dZAxisShow 29, 152
XtNxrt3dZAxisTitle 152
XtNxrt3dZGridLines 33, 153
XtNxrt3dZMax 29, 153
XtNxrt3dZMaxUseDefault 29, 153
XtNxrt3dZMin 29, 154
XtNxrt3dZMinUseDefault 29, 154
XtNxrt3dZoneData 18, 44, 155
  use of as 4D bar chart 88
  use of for 4D graphs 87
XtNxrt3dZoneMethod 53, 155
XtNxrt3dZOrigin 44, 48, 156
XtNxrt3dZRotation 76, 154
XtNxrt3dZScale 31, 155
XtNxrt3dZTitle 29
XtNxrt3dZValueLabels 155
XtNxrtBackgroundColor 36
XtNxrtColorFunc 37
XtNxrtDataAreaBackgroundColor 36
XtNxrtDataStyles 61
XtNxrtDataStylesUseDefault 58
XtNxrtForegroundColor 37

XtNxrtModifyCallback 74
XtNxrtRotateCallback 76
XtXrt3d 16, 133
XtXrt3dText 157
xtXrt3dWidgetClass 133
XtXrtGraph 16
XWD output 41, 178

Z

zone graphs 45
ZoomCancel() action 191
ZoomEnd() action 74, 149, 150, 191
zooming
  default translation 187
zooming, interactive 71
ZoomUpdate() action 190