XRT/field™ for Motif

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 2.0



Software Development Productivity

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

January 1999

Part No: KLFM20

Copyright © 1995-1999 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/field, XRT/graph) are licensed on a per toolkit (such as Motif) and per architecture (such as SPARC, RS6000, HP 700/800, OpenVMS AXP, etc.) basis.
- There are two types of XRT licenses: single and network-site.
- A single license authorizes use of an XRT product for development of end-user applications on a particular machine identified by a CPU ID. Development activities can include the following:
  - compiling and linking the XRT product library into an end-user application binary.
  - running the XRT Builder tool (if one is supplied).
  - integrating the XRT product into a GUI Builder or UIMS tool. Once an XRT product is integrated into such a tool, you may use the tool for XRT development only on machines that are licensed to use the XRT product.
- A network-site development license authorizes use of an XRT product for development of end-user applications on all similar-architecture machines on the same network and located in the same building.
- You may distribute end-user application binaries including the statically-linked XRT product library anywhere, and as many times as you like, without incurring royalty or run-time fees.

XRT FOR MOTIF BINARY LICENSE AGREEMENT

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

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

SOFTWARE LICENSE

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

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

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

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

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

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

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

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

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

LIMITED WARRANTY

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

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

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

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

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

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

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

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

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

Table of Contents

Preface 1
Introduction 1
New Features In XRT/field 2.0 2
Assumptions 3
Typographical Conventions Used in this Manual 3
Related Documents 3
Technical Support 3
Product Feedback and Announcements 4

Part I: Using XRT/field

1 Getting Started: Simple XRT/field Programs 9
1.1 Environment Variables 9
1.2 A Basic Text Field 10
1.3 Compiling and Authorizing Your Program 11
1.4 Performing Basic Validation 12
1.5 Creating a Combo Box or Spin Box 13
Creating a Combo Box 14
Creating a Spin Box 15
1.6 Creating an XrtCalendar Widget 17
1.7 Proceeding From Here 18

2 XRT/field Basics 19
2.1 Motif Integration 19
2.2 The XrtField Subclasses 21
2.3 Default User Interaction 21
2.4 Storing Values in Fields 23
2.5 Edit and Display Formats 24
2.6 Conversion, Completion and Validation 24
2.7 The Editing Process 24
Enter Field Action Processing 25
Keystroke Action Processing 25

Paste Action Processing 25
User Exit Action Processing 26
Programmatic Change Action Processing 27
The Complete Editing Process 28
2.8 Pointer Resources 29
2.9 Floating Point Resources 29
2.10 Colors 29
2.11 Resource Files and GUI Builders 30

3 Programming XRT/field 31
3.1 Resource Summary 31
3.2 Edit Format Specification 36
3.3 Specifying Date Edit Formats 38
3.4 Validation 40
Number of Lines 41
Line Length 41
Valid and Invalid Characters 41
Validation Masks 42
Data Type 46
Value Range 46
List of Valid Values 46
Validation Callbacks 48
3.5 Handling Invalid Fields 49
3.6 Conversion 51
3.7 Completion 52
3.8 Display Format Specification 53
3.9 Exit Callbacks 56
3.10 Field Colors 57
3.11 Specifying Multi-Line Text 58

4 Advanced XRT/field Programming. 59
4.1 Default Field Value 59
4.2 Enter Prompts and Enter Callbacks 61
4.3 Using XRT/field with XRT/table 61
4.4 Auto Matching Completion 64
4.5 Drop-Down List Creation 65
4.6 XrtComboBox Programming 65
4.7 XrtSpinBox Programming 70
4.8 XrtCalendar Programming 72
4.9 Date Editor Popup 75
4.10 Binding Actions to Keys 76

The Calendar Action . . . 76
The Cancel Action . . . 77
The Complete Action . . . 77
The Menu Action . . . 78
The Overstrike Action . . . 79
The Scrollbars Action . . . 79
The Spin Action . . . 80
4.11 Checking Multiple Fields . . . 81
4.12 Date Conversion . . . 82
4.13 Controlling Scrolling . . . 84
4.14 Defining Mask Characters . . . 85
4.15 XrtField and XmText Callbacks . . . 85
4.16 Using C++ with XRT/field . . . 87
4.17 Using UIL with XRT/field . . . 87
4.18 Converting Resource Values to Strings . . . 87
4.19 Internationalization . . . 88

Part II: Reference Appendices

A Resource Reference . . . 93
A.1 Widget Synopses . . . 93
A.2 XrtField Resources . . . 95
A.3 XmText Resource Summary . . . 117
A.4 Inherited XrtField Resources . . . 120
A.5 XrtComboBox Resources . . . 121
A.6 XrtSpinBox Resources . . . 122
A.7 Inherited XrtComboBox and XrtSpinBox Resources . . . 124
A.8 XrtCalendar Resources . . . 125

B Procedures and Methods Reference . . . 129

C Macros . . . 139

D Translations and Actions Reference . . . 141
D.1 Default Translations . . . 141
D.2 Action Routines . . . 142

E Data Types . . . 145

F Resource Types and Classes . . . 153
F.1 Common XrtField Resources . . . 153

F.2 XrtStringField Resources 155
F.3 XrtIntField Resources 155
F.4 XrtFloatField Resources 156
F.5 XrtDateField Resources 156
F.6 XrtComboBox Resources 157
F.7 XrtSpinBox Resources 157
F.8 XrtCalendar Resources 157

G Resource File and GUI Builder Usage 159
G.1 Syntax 159
G.2 Example Resource File 162

H UIL Arguments 167
H.1 Common XrtField Resources 167
H.2 XrtIntField Resources 168
H.3 XrtFloatField Resources 168
H.4 XrtDateField Resources 168
H.5 XrtComboBox Resources 169
H.6 XrtSpinBox Resources 169
H.7 XrtCalendar Resources 169
H.8 Reasons 170

I Sample Code 171
I.1 Examples 171
I.2 Demos 172

Index 173

Preface

Introduction ■ *New Features In XRT/field 2.0*
Assumptions ■ *Typographical Conventions Used in this Manual*
Related Documents ■ *Technical Support*
Product Feedback and Announcements

Introduction

The XRT/field Widget Family

Motif is composed of many different widgets such as buttons, sliders, and menus. Conceptually, XRT/field adds a new set of widgets for use in Motif version 1.2 or later. Writing programs using XRT/field is very similar to writing any other kind of Motif program; you now have more Motif widgets to work with.

XRT/field widgets store text and allow you to convert, complete and validate the text once it is stored. They provide an easy way to enter and verify data in Motif applications that require intensive data entry, such as database applications. Additional widgets allow you to create spin boxes and combo boxes that contain and manipulate XRT/field widgets. Each field widget has resources that determine how the field will look and behave.

The XRT/field widget family consists of four widget classes:

- XrtField and its subclasses, which store and manipulate text entered by the user.
- XrtComboBox, which allows users to choose an item from a pick list.
- XrtSpinBox, which can be used to increment numeric values or cycle through a pick list.
- XrtCalendar, which can be used as a date popup for a field or on its own within a program.

The XrtField widget class is defined as a subclass of the Motif XmText widget. This means that XRT/field widgets can be plugged directly into any application that uses text widgets, including applications that use KL Group's XRT/table widget. Similarly, the XrtComboBox, XrtSpinBox and XrtCalendar widgets are defined as subclasses of XmXrtFldManager, a subclass of the XmManager widget.

Features and Capabilities

The XrtField widget class provides the following capabilities:

- Validation: data can be validated using masks, data types, value ranges, or pick lists.
- Conversion: values can be converted using callback routines.
- Completion: partial values can be completed using callback routines or pick lists.
- Format specification: you can specify the edit and display format for the value.
- Enter, exit, validation, and error-detection callback routines can be used by applications.
- You can specify which input characters are to be treated as valid or invalid.
- Built-in scrollbars can be displayed as needed.

XrtField widgets also inherit the features of the XmText widget.

An XrtComboBox widget can control the placement and widget class of its child arrow widget.

New Features In XRT/field 2.0

The following list summarizes the new features introduced in XRT/field 2.0, and indicates where they are discussed in this manual.

- A new widget, XrtCalendar, has been added. This widget can be used on its own or as a date popup for an XrtDateField. Introductory information is located in "Creating an XrtCalendar Widget" on page 17, and more detailed information is available in "XrtCalendar Programming" on page 72 and "Date Editor Popup" on page 75.
- The date/time spinner function of the XrtSpinBox has been improved to allow spinning of subfields. See "Creating a Combo Box or Spin Box" on page 13 and "XrtCalendar Programming" on page 72.
- Auto matching completion has been added, allowing a user to only enter partial data to have complete data appear in a field. See "Auto Matching Completion" on page 64.
- The XrtComboBox resource has been improved to allow drop-down list creation simultaneously with the combo box's creation. See "Drop-Down List Creation" on page 65 for more details.
- A new action called Paste has been added, which consists of both drag and drop and clipboard paste into a masked field has been added. See "Paste Action Processing" on page 25.
- New and improved demos and examples.
- Bug fixes and minor improvements (see the *Release and Installation Notes* document for a listing of the main bugs fixed).

Assumptions

This manual assumes that you are proficient with the C language and the Motif toolkit. C concepts such as "an array of `char *`", and "pointer to a structure" must be understood before beginning to program Motif applications, including those that use XRT/field. An understanding of basic Motif concepts such as getting and setting resource values is required before continuing with this manual. See "Related Documents" on 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/field and Motif resource names and constants. ■ Commands that you enter at a terminal window.
Italic Text	■ Pathnames, filenames, program, procedure and parameter names. ■ New terms as they are introduced, and to emphasize important words. ■ Figure and table titles. ■ The names of other documents referenced in this manual, such as the *Motif Programming Manual*.

Related Documents

- *Motif Programming Manual* by Dan Heller and Paula M. Ferguson, O'Reilly & Associates, Inc.
- *Motif Reference Manual* by Paula M. Ferguson, O'Reilly & Associates, Inc.
- *X Toolkit Intrinsics Reference Manual*, *Xlib Programming Manual* and *Xlib Reference Manual* from O'Reilly & Associates, Inc. These manuals are not required to develop applications using XRT/field and Motif; however, an understanding of the Xt Intrinsics and 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/field. You may purchase annual XRT/care agreements from your XRT Reseller or KL Group. XRT/care ensures you will receive all XRT/field 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/field from KL Group, contact KL Group for technical support; if you purchased XRT/field 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/field 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/field 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 Resouces

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/field, 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/field

1

Getting Started: Simple XRT/field Programs

Environment Variables ■ *A Basic Text Field*
Compiling and Authorizing Your Program ■ *Performing Basic Validation*
Creating a Combo Box or Spin Box ■ *Creating an XrtCalendar Widget*
Proceeding From Here

This chapter allows you to immediately try out the XRT/field widgets by:

- Compiling and running a program that displays text in a string field.
- Changing the widget to an integer field and performing simple validation of the field's contents.
- Modifying the program to create first a combo box and then a spin box containing the field.
- Quickly implement code for a simple calendar widget.

1.1 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/field distribution files.

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

```
echo $XRTHOME
```

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

Before attempting to use XRT/field, 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/field.

1.2 A Basic Text Field

Figure 1 is an example that illustrates the use of XRT/field. This program, *simple.c*, creates a field of class XrtStringField, which stores a text string. (The field widget classes are described in more detail in section 2.2 on page 21.)

Most of the code should be familiar to Motif programmers. Lines 13-15 initialize the Intrinsics and create a toplevel shell in the standard Xt Intrinsics fashion. Lines 18-21 use the standard Xt Intrinsics procedure *XtVaCreateManagedWidget()* to instantiate the field.

The `XmNxrtFldInternalValue` resource stores the value for this field. In this case, the value is the character string "`this is my text`".

Standard Xt procedures (like *XtSetValues()*, *XtGetValues()*, *XtVaSetValues()*) may be used to manipulate the field.

This program is included as part of the XRT/field distribution, in the file *$XRTHOME/src/field/examples/simple.c*.

```
1 #include <X11/Intrinsic.h>
2 #include <Xm/Xm.h>
3 #include <Xm/Text.h>
4 #include <Xm/XrtField.h>
5
6 int
7 main (int argc, char *argv[])
8 {
9               XtAppContext            app_context;
10              Widget                  field, toplevel;
11
12              /* Initialize Intrinsics */
13              toplevel = XtVaAppInitialize(&app_context,
14                      "Simple", NULL, 0,
15                      &argc, argv, NULL, NULL);
16
17              /* Create field */
18              field = XtVaCreateManagedWidget("field",
19                      xmXrtStringFieldWidgetClass,    toplevel,
20                      XmNxrtFldInternalValue,         "this is my text",
21                      NULL);
22
23              XtRealizeWidget(toplevel);
24              XtAppMainLoop(app_context);
25 }
```

Figure 1 simple.c: a program that creates a simple field

1.3 Compiling and Authorizing Your Program

To compile and link the *simple.c* program, run the following commands:[1]

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

$XRTHOME/bin/xrt_auth simple
```

You can then run your program by entering the *simple* command. The field that appears on the screen is shown in Figure 2.

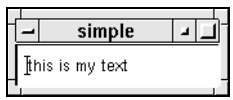


Figure 2 The field created by simple.c

Libraries and Include Files

The above commands assume that the X and Motif libraries and include files are installed in the standard places. Normally, these are */usr/lib*, */usr/include/X11*, and */usr/include/Xm*.

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

xrt_auth

After the XRT/field 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/field, and will enable the application. *xrt_auth* takes two optional switches, `-v` for verbose messages and `-q` for quiet mode (that is, no messages):

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

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

If your application is not successfully enabled using *xrt_auth*, XRT/field widgets will appear, but will be blank. The message shown in Figure 3 is written to *stdout*.

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

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

Figure 3 Message displayed by expired or non-authorized application

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

C++ and ANSI C

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

1.4 Performing Basic Validation

One of the most useful features of XRT/field is the ability to define rules that specify whether the value contained in a field is valid.

Figure 4 provides a simple example of a program that creates a field and validates its value. The differences between this program and the *simple.c* program you have already seen are shown in boldface.

This program differs from *simple.c* in two ways:

- The created field belongs to the XrtIntField (integer) widget class. These widgets only accept integers; values that are not integers are flagged as invalid. (For an explanation of field widget classes, see section 2.2 on page 21.)
- The `XmNxrtFldMin` and `XmNxrtFldMax` resources are set to define minimum and maximum field values. Values smaller than 1 or larger than 10 are flagged as invalid.

```
1 #include <X11/Intrinsic.h>
2 #include <Xm/Xm.h>
3 #include <Xm/Text.h>
4 #include <Xm/XrtField.h>
5
6 int
7 main (int argc, char *argv[])
8 {
9               XtAppContext            app_context;
10              Widget                  field, toplevel;
11
12              /* Initialize Intrinsics */
13              toplevel = XtVaAppInitialize(&app_context,
14                      "Simple", NULL, 0,
15                      &argc, argv, NULL, NULL);
16
17              /* create field */
18              field = XtVaCreateManagedWidget("field",
19                      xmXrtIntFieldWidgetClass,       toplevel,
20                      XmNxrtFldMin,                   1,
21                      XmNxrtFldMax,                   10,
22                      NULL);
23
24              XtRealizeWidget(toplevel);
25              XtAppMainLoop(app_context);
26 }
```

Figure 4 Modifications to simple.c to include an example of validation

Other Validation Methods

XRT/field provides the following validation methods, in addition to the ones shown above:

- Field width and length.
- List of valid or invalid characters.
- Validation mask.
- List of valid values (pick list).
- Validation callbacks.

Section 3.4 on page 40 describes each method in detail.

1.5 Creating a Combo Box or Spin Box

Fields are usually created as children of other widgets. For example, several fields may be contained in an XmForm widget.

For convenience, XRT/field provides two composite widgets that contain a field:

- XrtComboBox, which creates and manages a field and an arrow widget.

- XrtSpinBox, which creates and manages a field and two arrow widgets.

1.5.1 Creating a Combo Box

The following program modifies *simple.c* to create an XrtComboBox widget. Modifications are shown in boldface.

```
1 #include <X11/Intrinsic.h>
2 #include <Xm/Xm.h>
3 #include <Xm/Text.h>
4 #include <Xm/XrtField.h>
5
6 int
7 main (int argc, char *argv[])
8 {
9               XtAppContext            app_context;
10              Widget                  toplevel, combobox;
11              static String picklist[] = {"John Jones","Harry Smith",
12                      "Jean Jones","Mary Martin","Harry Hayes",NULL};
13
14              /* Initialize Intrinsics */
15              toplevel = XtVaAppInitialize(&app_context,
16                      "Combo", NULL, 0,
17                      &argc, argv, NULL, NULL);
18
19              /* Create combo box */
20              combobox = XtVaCreateManagedWidget("combobox",
21                      xmXrtComboBoxWidgetClass,       toplevel,
22                      XmNxrtComboFieldType,           XRTFLD_FIELDTYPE_STRING,
23                      XmNxrtFldPickList,              picklist,
24                      NULL);
25
26              XtRealizeWidget(toplevel);
27              XtAppMainLoop(app_context);
28 }
```

Figure 5 simple.c modified to create an XrtComboBox widget

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

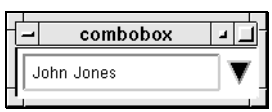


Figure 6 The XrtComboBox widget created by this program

In this program, `XmNxrtFldPickList`, a field resource, defines a list of legal values for the field. (You can set field resources when defining a combo box. You can also use *XtVaSetValues()* and *XtVaGetValues()* to set and retrieve field resources from within the combo box.)

The field subclass to use is specified by the `XmNxrtComboFieldType` resource. `XRTFLD_FIELDTYPE_STRING` specifies an XrtStringField widget.

When the user clicks on the arrow widget, a popup menu (of class XmList) appears, which contains the list of legal values defined by `XmNxrtFldPickList`:

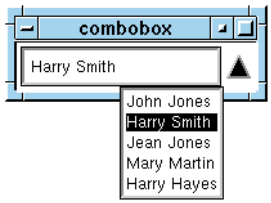


Figure 7 A popup menu contained in an XrtComboBox widget

The user can select an item from this list by clicking on it. Once the user has selected an item, the popup menu disappears and the field contains the selected value. The user can make the menu disappear without changing the field by clicking on the arrow key again.

For more information on programming with the XrtComboBox widget, refer to section 4.6 on page 65.

1.5.2 Creating a Spin Box

Creating an XrtSpinBox widget is similar to creating an XrtComboBox widget. Figure 9 adapts *simple.c* to create an XrtSpinBox widget. When it is compiled and run, the window in Figure 8 is displayed.

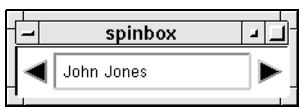


Figure 8 The XrtSpinBox widget created by spin.c

When an XrtSpinBox widget is created, three children are automatically created with it: a field and two arrow widgets. When the user clicks on the right arrow button, the field displays the next element of the pick list (cycling back to the beginning if necessary). The user can also go backwards in the pick list by clicking on the left arrow button.

You can assign values to the child field at the time the XrtSpinBox widget is created. As with XrtComboBox, you can also use *XtVaSetValues()* and *XtVaGetValues()* to set and retrieve field resources from within the box.

XrtSpinBox widgets can also be used as counters. In this case, when the user presses the right arrow button, the value contained in the widget increases. When the user presses the left arrow button, the widget decreases. For more information on counters and other features of XrtSpinBox widgets, see section 4.7 on page 70.

```
1 #include <X11/Intrinsic.h>
2 #include <Xm/Xm.h>
3 #include <Xm/Text.h>
4 #include <Xm/XrtField.h>
5
6 int
7 main (int argc, char *argv[])
8 {
9               XtAppContext            app_context;
10              Widget                  toplevel, spinbox;
11              static String picklist[] = {"John Jones","Harry Smith",
12                      "Jean Jones","Mary Martin","Harry Hayes",NULL};
13
14              /* Initialize Intrinsics */
15              toplevel = XtVaAppInitialize(&app_context,
16                      "Spin", NULL, 0,
17                      &argc, argv, NULL, NULL);
18
19              /* Create spin box */
20              spinbox = XtVaCreateManagedWidget("spinbox",
21                      xmXrtSpinBoxWidgetClass,        toplevel,
22                      XmNxrtSpinFieldType,            XRTFLD_FIELDTYPE_STRING,
23                      XmNxrtFldPickList,              picklist,
24                      NULL);
25
26              XtRealizeWidget(toplevel);
27              XtAppMainLoop(app_context);
28 }
```

Figure 9 simple.c modified to create an XrtSpinBox widget

1.6 Creating an XrtCalendar Widget

Creating an XrtCalendar widget is similar to creating an XrtSpinBox or XrtComboBox widget. Figure 10 shows the code required to create the XrtCalendar widget shown in Figure 11.

```
1  #include <Xm/Xm.h>
2  #include <Xm/XrtField.h>
3
4  int
5  #ifndef _NO_PROTO
6  main (int argc, char *argv[])
7       #else
8       main(argc, argv)
9       int argc;
10      char *argv[];
11      #endif
12      {
13              XtAppContext app_context;
14              Widget toplevel, calendar;
15
16              /* Initialize Intrinsics */
17              toplevel = XtVaAppInitialize(&app_context, "Calendar", NULL, 0,
18                      &argc, argv, NULL, NULL, 0);
19
20              /* Create calendar */
21              calendar = XtVaCreateManagedWidget("calendar",
22                      xmXrtCalendarWidgetClass, toplevel,
23                      NULL);
24
25              XtRealizeWidget(toplevel);
26              XtAppMainLoop(app_context);
27 }
```

Figure 10 Minimal XrtCalendar widget code example

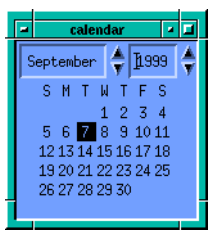


Figure 11 A simple XrtCalendar widget

The XrtCalendar widget is a compound widget consisting of three optional spin boxes with date fields and a day grid. By default, the time spin box is not shown, but is positioned at the bottom of the calendar display when it is enabled. The spin boxes

for the month, year and time all function exactly the same as all other XrtSpinBox widgets. See “Creating a Combo Box or Spin Box” on page 13.

The user can specify a maximum and minimum roll date (range of dates) for the widget to restrict date selection. Also, by using `XmNxrtCalendarEnabled`, you can depict whether user interaction with the widget is allowed at all or not.

The user can move from field to field using the TAB key, and can move within the day grid from day to day using the arrow keys. The time and year field can be edited manually (numbers can be typed directly in the field by the user) or by using the spin box controls, but the month field can only be altered with the spin box controls. If the calendar is a date popup, pressing the ENTER key dismisses the calendar using the `Calendar(Hide)` action. See Appendix D on page 141 for more information.

As with XrtComboBox and XrtSpinBox, you can also use *XtVaSetValues()* and *XtVaGetValues()* to set and retrieve field resources from within the calendar. For more information on other features of XrtCalendar widgets, see section 4.8 on page 72.

1.7 Proceeding From Here

The following suggestions should help you be productive with XRT/field as quickly as possible:

- Chapter 2 contains basic XRT/field concepts. You should read this chapter to get a good grounding in XRT/field.
- The examples and demos are useful learning aids. You can see what XRT/field can do, and how to do it, by running them and examining their source code. Appendix I on page 171 briefly describes all of the demos and examples.

2

XRT/field Basics

Motif Integration ■ *The XrtField Subclasses*
Default User Interaction ■ *Storing Values in Fields*
Edit and Display Formats ■ *Conversion, Completion and Validation*
The Editing Process ■ *Pointer Resources*
Floating Point Resources ■ *Colors*
Resource Files and GUI Builders

XRT/field defines widgets that validate, convert and complete entered data. Different widgets handle different types of data. SpinBox, ComboBox and Calendar widgets can be defined to manipulate the data.

2.1 Motif Integration

Conceptually, XRT/field adds four new widget classes to the Motif toolkit:

- XrtField, which defines subclasses that extend the XmText widget to provide additional conversion, validation and display capabilities.
- XrtComboBox, a combo box created with an XrtField widget.
- XrtSpinBox, a spin box created with an XrtField widget.
- XrtCalendar, a calendar widget.

Motif programmers manipulate these widgets using the familiar Xt Intrinsics calls.

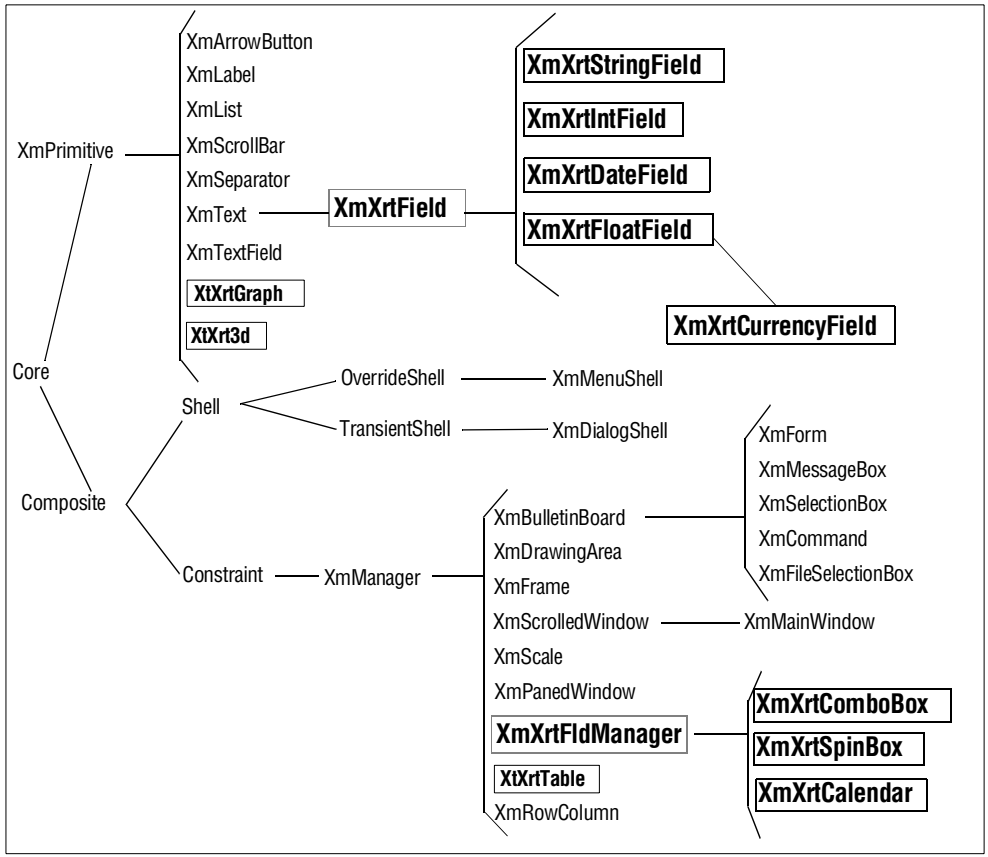


Figure 12 XRT/field widgets' position within the Motif Class Hierarchy

The Xt Intrinsics defines an object-oriented architecture that organizes widgets into classes. For example, the Motif XmForm widget is a subclass of XmBulletinBoard, which is a subclass of XmManager, which is subclassed from the Constraint, Composite and Core widgets.

The XrtField widget class is defined as a subclass of XmText, which is in turn subclassed from XmPrimitive. This means that XmText, XmPrimitive and Core resources can be set in addition to the resources defined by the XrtField widget. For example, you can set the default background color with the `XmNbackground` resource.

XrtComboBox, XrtSpinBox and XrtCalendar are subclassed from XmXrtFldManager.

Replacing XmText Widgets

As a subclass of XmText, any XrtField widget can replace any XmText widget in an application.

Replacing XmTextField Widgets

You should be able to replace XmTextField widgets with XrtField widgets without changing the application's behavior, since XmTextField has the same resources as XmText.

2.2 The XrtField Subclasses

XrtField is an abstract class that defines several subclasses. You cannot instantiate the XrtField class. Each of the subclasses (shown in Figure 13) handles a certain type of user input.

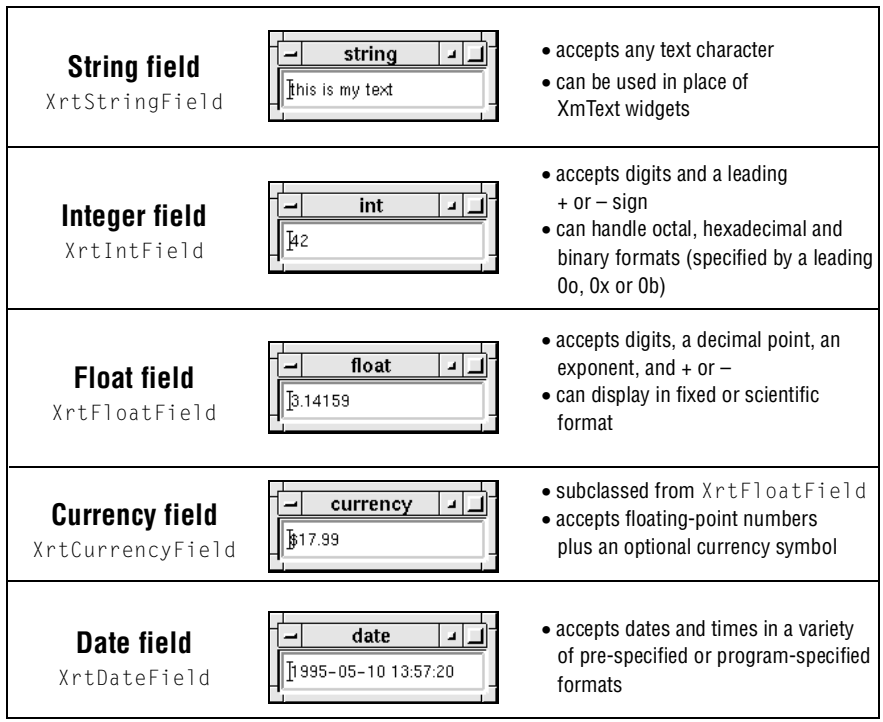


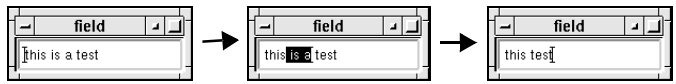
Figure 13 The subclasses of the XrtField widget

In this document, the term *field* refers to a widget that belongs to one of the subclasses of XrtField.

2.3 Default User Interaction

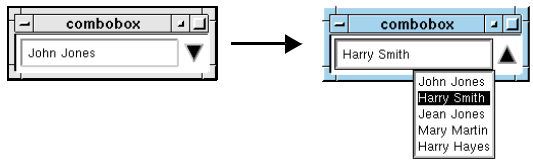
Figure 14 illustrates the most common ways users interact with XRT/field widgets. For a complete list of user interactions, refer to the translation tables in Appendix D on page 141.

XrtField subclasses



- Enter or change text using standard XmText controls
- Press Tab or Return to traverse out of the field

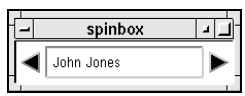
XrtComboBox



MOUSE:
- Click on arrow button to bring up popup menu
- Click on arrow button again to dismiss the popup menu, or press osfCancel or click outside of menu
- Click on item in popup menu to change the value stored in the XrtField widget

KEYBOARD:
- Press Ctrl-Down to bring up pop-up menu (Ctrl-Up dismisses menu)
- Move up and down in the popup menu by pressing the Up and Down keys
- Move to a specific item in the popup menu by typing its first character
- Press Return to select an item from the popup menu

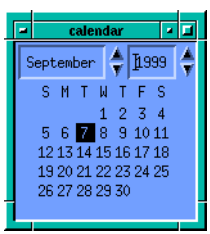
XrtSpinBox



MOUSE:
- Click on the right arrow button to display the next picklist item
- Click on the left arrow button to display the previous picklist item

KEYBOARD:
- Press Up to display the next item
- Press Down to display the previous item

XrtCalendar



MOUSE:
- Click on the up arrow buttons to increase the current value
- Click on the down arrow buttons to decrease the current value
- Click on a day in the daygrid to make that the currently selected date

KEYBOARD:
- Press Tab to move between fields
- When daygrid has the focus, use arrow keys to move around the grid
- When month, year or time fields are current, use up or right arrow keys to increase the current value
- When month, year or time fields are current, use down or left arrow keys to decrease the current value

Figure 14 XRT/field User Interactions

2.4 Storing Values in Fields

To store a value in a field, you can set either of two resources: `XmNxrtFldInternalValue` or `XmNvalue`, though for optimal XRT functionality, it is best to use `XmNxrtFldInternalValue`. The exception is when you want to set visual cues for a field (for example, have *dd/mm/yy* display in the field so the user knows what can be entered where). In this case, you need to use `XmNvalue`.

Using XmNxrtFldInternalValue

The `XmNxrtFldInternalValue` resource stores the value using the data type appropriate for the widget subclass.

Field Subclass	Internal Value Type
XrtStringField	string
XrtIntField	long
XrtFloatField	double
XrtCurrencyField	double
XrtDateField	struct tm

For example, the following creates an integer field and uses `XmNxrtFldInternalValue` to set its value:

```
field = XtVaCreateManagedWidget("field",
    xmXrtIntFieldWidgetClass,   toplevel,
    XmNxrtFldInternalValue,     42,
    NULL);
```

When the user exits a field, the value typed by the user is copied from `XmNvalue` to `XmNxrtFldInternalValue` (with type conversion performed when necessary).

XmNxrtFldInternalValueLong

If the widget is a date field, the `XmNxrtFldInternalValueLong` resource contains, if possible, the value of `XmNxrtFldInternalValue` represented as an integer of type *long*. (This is only possible if the date is in a range roughly between the years 1902 and 2033.)

Using XmNvalue

`XmNvalue` contains the actual text string stored and displayed in the field. The following uses `XmNvalue` to set the initial value of an integer field:

```
field = XtVaCreateManagedWidget("field",
    xmXrtIntFieldWidgetClass,   toplevel,
    XmNvalue,                   "42",
    NULL);
```

If `XmNvalue` is set to a value inappropriate for the widget's subclass (such as "`abc`" for an integer field), the value is displayed but is flagged as invalid.

Note: The only time `XmNvalue` needs to be used is in the context of setting visual cues in a masked field. See "Specifying Visual Cues" on page 45.

2.5 Edit and Display Formats

XRT/field allows you to specify two formats that control the appearance of the value of a field:

- The *edit format*, which controls the appearance of a value while it is being edited.
- The *display format*, which controls the appearance of a value while it is not being edited.

For details on edit formats, see section 3.2 on page 36. For information on display formats, see section 3.8 on page 53.

2.6 Conversion, Completion and Validation

XRT/field allows you to manipulate the field's value during the editing process in three ways:

- You can *convert* the value from one form to another. For example, the value can be converted from lower-case to upper-case.
- You can *complete* the value by adding a prefix or suffix to it. For example, you can add a *.c* suffix to a file name.
- You can *validate* the value.

These operations can be performed at various points in the field editing process. For details, see the following section.

2.7 The Editing Process

The field editing process can be broken down into the following steps:

- The user first enters the widget (gives the field input focus): this is called an *enter field action*.
- The user types one or more keystrokes. Each keystroke typed is called a *keystroke action*.
- The user performs a paste or drag and drop operation: this is called a *paste action*.
- Finally, the user exits the widget by pressing Return or Tab, or by moving the mouse pointer out of the widget: this is called a *user exit action*.

The application can itself modify the value stored in the field. A modification not initiated by a user is known as a *programmatic change action*.

XRT/field allows you to add processing after any of these actions. The sequence of interactions and possible processing is shown in Figure 15.

2.7.1 Enter Field Action Processing

When an enter field action occurs, you can provide the following:

- A list of callback routines to be invoked following an enter field action. For details on how to specify enter callbacks, see section 4.2 on page 61.
- An edit format that determines how the value is to be displayed while being edited. For more information on edit format specification, refer to section 3.2 on page 36.

2.7.2 Keystroke Action Processing

When a keystroke action occurs, you can do the following:

- Convert the (partially typed) value from one format to another. For details on the conversion capabilities of XRT/field, refer to section 3.6 on page 51.
- Determine whether the typed key is valid. For information on validation, refer to section 3.4 on page 40.

2.7.3 Paste Action Processing

When a paste action occurs, you can do the following:

- Convert the pasted value from one format to another. For details on the conversion capabilities of XRT/field, refer to section 3.6 on page 51.
- Determine whether the pasted value is valid. For information on validation, refer to section 3.4 on page 40.

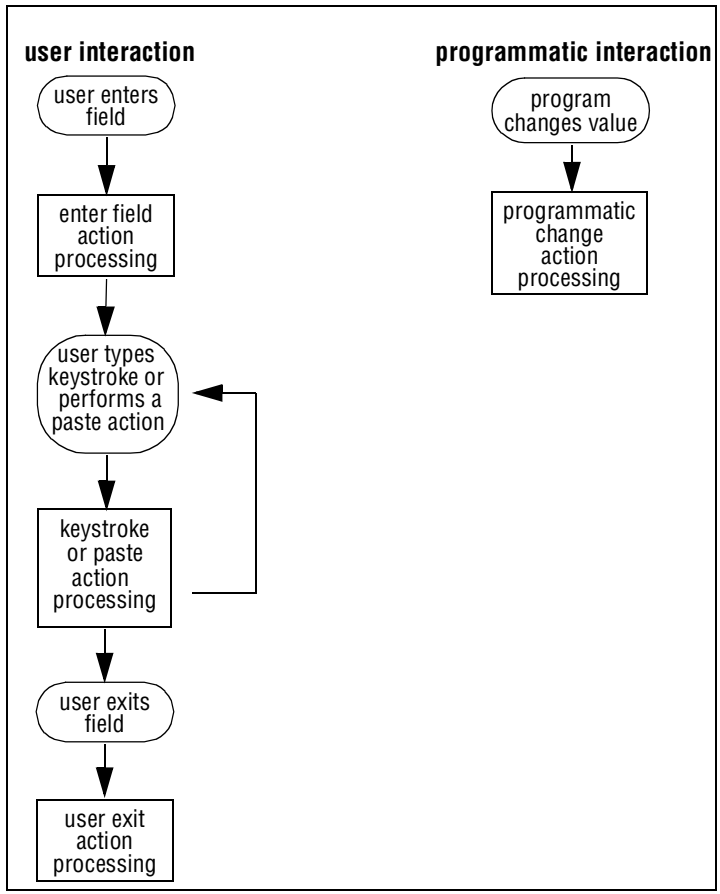


Figure 15 Interactions and the resulting processing

2.7.4 User Exit Action Processing

When the user exits the field, a user exit action occurs. A "user exit" is defined to be any of the following user activities:

- Moving the mouse pointer to another window while `XmNfocusPolicy` is set to `XmPOINTER`.
- Attempting to traverse to the next widget in the window by pressing Tab.
- Pressing Return. (This is only supported if the field does not accept multi-line input. See section 3.11 on page 58 for details on multi-line input support.)

Note that, unlike the XmText widget, field widgets traverse to the next widget when Return is pressed. To find out how to disable this behavior, see section 3.9 on page 56.

When a user exit action occurs, you can do the following:

- Convert the value from one format to another. For details on the conversion capabilities of XRT/field, refer to section 3.6 on page 51.
- Complete any incompletely typed value. For information on completion, refer to section 3.7 on page 52.
- Determine whether the value is valid. For information on validation, refer to section 3.4 on page 40.
- Specify the format to use when displaying the value. For details on display format specification, refer to section 3.8 on page 53.
- Specify exit callbacks to be invoked. For details on these callbacks, see section 3.9 on page 56.

2.7.5 Programmatic Change Action Processing

When the value stored in a field is changed by the program itself, a programmatic change action occurs. The following are examples of programmatic change actions:

- The program changes the value of `XmNvalue` or `XmNxrtFldInternalValue`.
- An XmText procedure is called.
- A value is selected from the popup menu displayed by the *Menu()* action routine. (See section 4.10.4 on page 78 for a discussion of *Menu()*.)
- `XmNvalue` or `XmNxrtFldInternalValue` is set at the time of widget creation. (This ensures that the initial value of a field is validated.)

When a programmatic change action occurs, you can do the following:

- Convert the current value of the field from one format to another. For details on the conversion capabilities of XRT/field, refer to section 3.6 on page 51.
- Determine whether the value is valid. For information on validation, refer to section 3.4 on page 40.
- Specify the format to use when displaying the value. For details on display format specification, refer to section 3.8 on page 53.

2.7.6 The Complete Editing Process

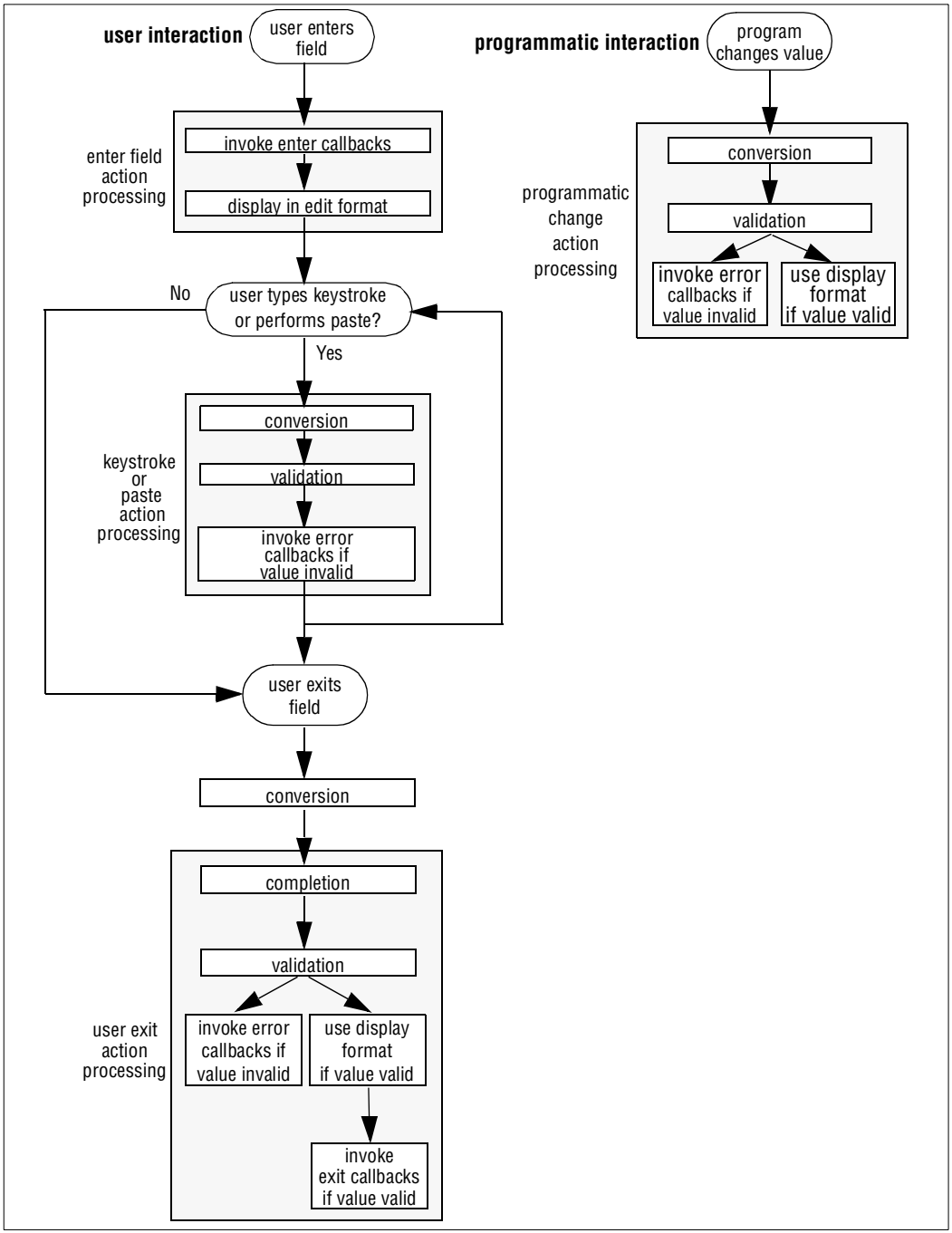


Figure 16 The complete XrtField editing process

Figure 16 shows the complete field editing process. This diagram may seem complex: however, it is just Figure 15 expanded to list the features described above and the order in which they are performed.

2.8 Pointer Resources

Some XRT/field resources return pointers to structures or strings when used in an *XtGetValues()* call. Since this is a pointer to the widget's internal storage, an application should *never* use this pointer to change any data in memory, and the pointer should never be altered or freed. Everything the pointer points to should be considered *read only* and *temporary*. (The exception is `XmNvalue`, which is copied. The value of this resource should be freed after use.)

XRT/field makes its own internal copy of all data passed to it through a pointer resource.

Xt procedures such as *XtNewString()* can be used to make copies of some types of data, and Xt procedures such as *XtFree()* can be used to release these copies.

2.9 Floating Point Resources

When setting floating-point resources in float or currency fields, you must use the *XrtFldDoubleToArgVal()* procedure. For example, to set the maximum floating-point value that can be stored in a field:

```
double maxval;
maxval = 125.5;
XtVaSetValues(myfield,
    XmNxrtFldMax, XrtFldDoubleToArgVal(maxval),
    NULL);
```

2.10 Colors

When an application built with XRT/field is displayed on a monochrome X Server:

- All non-black colors appear as white when used for background colors.
- All other non-white colors appear as black.

2.11 Resource Files and GUI Builders

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.

Using resource files, you can set all of XRT/field's resources (except for callback resources). Resource files make it easy to prototype different resource settings and quickly see the results. Appendix G on page 159 provides details on the syntax of resource settings when used in resource files.

Using GUI Builders

XRT/field integrates with popular GUI builders. To set XRT/field resources from a GUI builder, use the same syntax as that used in resource files. For information on this syntax, see Appendix G on page 159.

3

Programming XRT/field

Resource Summary ■ *Edit Format Specification*
Specifying Date Edit Formats ■ *Validation*
Handling Invalid Fields ■ *Conversion*
Completion ■ *Display Format Specification*
Exit Callbacks ■ *Field Colors*
Specifying Multi-Line Text

3.1 Resource Summary

The following figures list all of the XRT/field resources. It is not necessary to remember all the resources to program XRT/field effectively. For most widgets, resources may be left at their default settings.

Resource	Type
XmNxrtFldAllowNull	Boolean
XmNxrtFldAutoComplete	Boolean
XmNxrtFldCase	XrtFldCase
XmNxrtFldCompleteCallback	XtCallbackList
XmNxrtFldConvertCallback	XtCallbackList
XmNxrtFldCreatePopups	Boolean
XmNxrtFldDefaultBackground	Pixel
XmNxrtFldDefaultForeground	Pixel
XmNxrtFldDisplayFormat	XrtFldFormat
XmNxrtFldDisplayHorizScrollBar	XrtFldDispScrollBar
XmNxrtFldDisplayVertScrollBar	XrtFldDispScrollBar
XmNxrtFldEditFormat	XrtFldFormat
XmNxrtFldEnterCallback	XtCallbackList

continued on next page...

Resource	Type
XmNxrtFldErrorBackground	Pixel
XmNxrtFldErrorCallback	XtCallbackList
XmNxrtFldErrorForeground	Pixel
XmNxrtFldErrorString	XmString
XmNxrtFldExitCallback	XtCallbackList
XmNxrtFldInvalidChars	String
XmNxrtFldIsDefault	Boolean
XmNxrtFldIsModified	Boolean
XmNxrtFldIsValid	Boolean
XmNxrtFldLastError	XrtFldError
XmNxrtFldMask	String
XmNxrtFldMaskChars	String
XmNxrtFldMatchMask	XrtFldMatchMask
XmNxrtFldMatchPickList	Boolean
XmNxrtFldMaxLineLength	int
XmNxrtFldMaxLines	int
XmNxrtFldMenuAlignment	XrtFldMenuAlignment
XmNxrtFldMenuCallback	XtCallbackList
XmNxrtFldMenuList	Widget
XmNxrtFldOverstrike	Boolean
XmNxrtFldPickListIndex	int
XmNxrtFldPopupType	XrtFldPopupType
XmNxrtFldPromptString	XmString
XmNxrtFldRange	XrtFldRange
XmNxrtFldSearchPolicy	XrtFldSearchPolicy
XmNxrtFldSpinCallback	XtCallbackList
XmNxrtFldValidateCallback	XtCallbackList
XmNxrtFldValidChars	String
XmNxrtFldVisibleItemCount	int

Figure 17 Resources common to all XrtField widgets

Resource	Type
XmNxrtFldDefaultValue	String
XmNxrtFldInternalValue	String
XmNxrtFldMax	String
XmNxrtFldMin	String
XmNxrtFldPickList	String *

Figure 18 Additional XrtStringField Resources

Resource	Type
XmNxrtFldCounterIncrement	long
XmNxrtFldDefaultValue	long
XmNxrtFldDisplayFormatString	String
XmNxrtFldInternalValue	long
XmNxrtFldMax	long
XmNxrtFldMin	long
XmNxrtFldNegativeFormat	XrtFldNegFormat
XmNxrtFldPickList	long *
XmNxrtFldRange	XrtFldRange

Figure 19 Additional XrtIntField Resources

Resource	Type
XmNxrtFldCounterIncrement	double
XmNxrtFldDecimalPlaces	int
XmNxrtFldDefaultValue	double
XmNxrtFldDisplayFormatString	String
XmNxrtFldInternalValue	double
XmNxrtFldLeadingZero	Boolean
XmNxrtFldMax	double
XmNxrtFldMin	double
XmNxrtFldNegativeFormat	XrtFldNegFormat
XmNxrtFldPickList	double *
XmNxrtFldRange	XrtFldRange

Figure 20 Additional XrtFloatField Resources

Resource	Type
XmNxrtFldCalendar	Widget
XmNxrtFldCalendarCallback	XtCallbackList
XmNxrtFldCounterIncrement	struct tm *
XmNxrtFldDefaultValue	struct tm *
XmNxrtFldDisplayFormatString	String
XmNxrtFldEnterFormatList	String *
XmNxrtFldInternalValue	struct tm *
XmNxrtFldInternalValueLong	long
XmNxrtFldMax	struct tm *
XmNxrtFldMilleniumThreshold	int
XmNxrtFldMin	struct tm *
XmNxrtFldSpinPolicy	XrtFldSpinPolicy
XmNxrtFldPickList	struct tm **

Figure 21 Additional XrtDateField Resources

Resource	Type
XmNxrtComboArrow	Widget
XmNxrtComboArrowClass	WidgetClass
XmNxrtComboArrowType	XrtFldArrowType
XmNxrtComboField	Widget
XmNxrtComboFieldClass	WidgetClass
XmNxrtComboFieldType	XrtFldFieldType
XmNxrtComboSpacing	Dimension

Figure 22 XrtComboBox Resources

Resource	Type
XmNxrtSpinArrowLayout	XrtSpinArrowLayout
XmNxrtSpinArrowOrientation	unsigned char
XmNxrtSpinArrowType	XrtFldArrowType
XmNxrtSpinDecrArrow	Widget
XmNxrtSpinDecrArrowClass	WidgetClass
XmNxrtSpinField	Widget
XmNxrtSpinFieldClass	WidgetClass
XmNxrtSpinFieldName	String
XmNxrtSpinFieldType	XrtFldFieldType
XmNxrtSpinIncrArrow	Widget
XmNxrtSpinIncrArrowClass	WidgetClass
XmNxrtSpinSpacing	Dimension
XmNxrtSpinType	XrtSpinType

Figure 23 XrtSpinBox Resources

Resource	Type
XmNxrtCalendarCellMargin	Dimension
XmNxrtCalendarCellHighlightThickness	Dimension
XmNxrtCalendarCreateSpinMonth	Boolean
XmNxrtCalendarCreateSpinTime	Boolean
XmNxrtCalendarCreateSpinYear	Boolean
XmNxrtCalendarModifyCallback	XtCallbackList
XmNxrtCalendarEnabled	Boolean
XmNxrtCalendarFirstDayOfWeek	int
XmNxrtCalendarFullDayNames	Boolean
XmNxrtCalendarHeaderNumChars	int
XmNxrtCalendarHeaderBackground	Pixel
XmNxrtCalendarHeaderFontList	XmFontList
XmNxrtCalendarHeaderForeground	Pixel
XmNxrtCalendarMarginBottom	Dimension
XmNxrtCalendarMarginLeft	Dimension
XmNxrtCalendarMarginRight	Dimension
XmNxrtCalendarMarginTop	Dimension
XmNxrtCalendarMaxRollDate	struct tm*
XmNxrtCalendarMinRollDate	struct tm*
XmNxrtCalendarSpecialDayBackground	Pixel

continued on next page...

Resource	Type
XmNxrtCalendarSpecialDayForeground	Pixel
XmNxrtCalendarSpecialDays	XrtCalendarSpecialDay **
XmNxrtCalendarSelectedBackground	Pixel
XmNxrtCalendarSelectedForeground	Pixel
XmNxrtCalendarValue	struct tm*
XmNxrtCalendarValueLong	long

Figure 24 XrtCalendar Resources

3.2 Edit Format Specification

When the user enters a field, the value of the field (stored in the XmNxrtFldInternalValue resource) is made available to the user for editing. To control how the field value appears while it is being edited, set the XmNxrtFldEditFormat resource.

The set of valid values for XmNxrtFldEditFormat depends on the subclass of the field.

Integer Fields

In integer fields, XmNxrtFldEditFormat can be set as follows:

XRTFLD_FORMAT_BINARY	display as a sequence of ones and zeroes
XRTFLD_FORMAT_GENERAL	display using printf %d (long decimal) format (the default)
XRTFLD_FORMAT_HEX	display in hexadecimal
XRTFLD_FORMAT_OCTAL	display in octal
XRTFLD_FORMAT_THOUSANDS	display in groups of three digits separated by the separation character for the current locale (for example, 200,000)

The user can override the edit format specified above by typing prefix characters before typing the value to be stored in the field. "0b" specifies binary input, "0o" specifies octal, and "0x" specifies hexadecimal.

Float and Currency Fields

In float fields, XmNxrtFldEditFormat can be set as follows:

XRTFLD_FORMAT_FIXED	fixed-precision format (equivalent to printf %f format)
XRTFLD_FORMAT_SCIENTIFIC	scientific notation (equivalent to printf %e format)

XRTFLD_FORMAT_GENERAL	display using printf %g format (the default; uses either %e or %f format, whichever is shorter)
XRTFLD_FORMAT_PERCENT	display the number as a percentage (for example, display .75 as 75%)
XRTFLD_FORMAT_THOUSANDS	display in groups of three digits separated by the separation character for the current locale (for example, 2,453.61)

In currency fields, XmNxrtFldEditFormat can be set to the float field values shown above, plus the following:

XRTFLD_FORMAT_CURRENCY	standard currency notation for the current locale (for example, $1,234.00)
XRTFLD_FORMAT_CURRENCY_INTL	international currency notation (for example, USD0 1,234.00)

The default value for currency fields is XRTFLD_FORMAT_FIXED.

The following example shows how to define a field whose value is to be displayed in scientific notation while being edited:

```
field = XtVaCreateManagedWidget("field",
    xmXrtFloatFieldWidgetClass, toplevel,
    XmNxrtFldInternalValue,
            XrtFldDoubleToArgVal(16857.0),
    XmNxrtFldEditFormat,        XRTFLD_FORMAT_SCIENTIFIC,
    XmNxrtFldDecimalPlaces,     4,
    NULL);
```

When the field is next edited, the value will be displayed as 1.6857e+04.

Specifying Decimal Places

The XmNxrtFldDecimalPlaces resource controls the number of digits displayed after the decimal point. The default value for XmNxrtFldDecimalPlaces is 2 if the field is a float field, or XRTFLD_LOCALE (the number of decimal places in the current locale) if the field is a currency field. XmNxrtFldDecimalPlaces is ignored if the edit format is XRTFLD_FORMAT_GENERAL.

If XmNxrtFldDecimalPlaces had not been explicitly set in the above example, the value 1.69e+04 (the internal value rounded to two decimal places) would have been displayed.

Editing Other Subclasses

Date fields use the first element of the resource XmNxrtFldEnterFormatList as their edit format. See section 3.3 on page 38 for information on XmNxrtFldEnterFormatList.

For string fields, the XmNxrtFldEditFormat resource has no meaning (since strings of characters are always displayed as is).

Edit versus Display Format

`XmNxrtFldEditFormat` only controls how the value is initially displayed when the field gets focus. The `XmNxrtFldDisplayFormat` resource controls how the value is displayed once the user has finished editing and the widget no longer has focus.

For details on `XmNxrtFldDisplayFormat`, refer to section 3.8 on page 53.

Creating Read-Only Fields

By default, fields can be edited by the user. To define a read-only field, set the `XmNeditable` and `XmNcursorPositionVisible` resources to `False`.

3.3 Specifying Date Edit Formats

One problem with entering dates and times is that their format is not always apparent. For example, the date "`01/10/94`" could mean either January 10, 1994 or October 1, 1994, depending on whether the day or month is listed first. Without additional information, it is impossible to tell which is correct.

The `XmNxrtFldEnterFormatList` resource allows you to deal with problems such as the above by defining a list of acceptable date formats for a particular date field. When a value is entered by the user, its format is determined by comparing it with the date formats in the list.

A date format is a character string containing some or all of the following:

D	numeric day of month (1-31)
DD	numeric day of month with leading zero
W	abbreviated weekday name
WW	weekday name
M	numeric month of year (1-12)
MM	numeric month of year with leading zero
MMM	abbreviated month name
MMMM	month name
Y or YY	year within current century (00-99)
YYYY	year including century
h	hour (0-23)
hh	hour with leading zero
m	minute (0-59)
mm	minute with leading zero
s	second (0-59)
ss	second with leading zero
A, P	AM/PM representation
Z	time zone abbreviation
\\	escape character (next character is a literal)

For instance, to interpret the date 01/10/94 as January 10, 1994, use the format MM/DD/YY. To treat it as October 1, 1994, use DD/MM/YY.

All characters not listed above are skipped during processing. To use any of the characters listed above as literals, precede the letter with two backslashes. For example, use \\M to indicate the letter M instead of the numeric month.

The following is an example of a field that uses XmNxrtFldEnterFormatList:

```
static String datelist[] = {"DD/MM/YY hh:mm:ss",
        "YYYY/MM/DD hh:mm:ss", NULL};
XtVaSetValues(field,
    XmNxrtFldEnterFormatList,   datelist,
    NULL);
```

Here, the acceptable date formats are DD/MM/YY hh:mm:ss and YYYY/MM/DD hh:mm:ss.

The Default Date Format List

If XmNxrtFldEnterFormatList is not set, the following default list of date formats is assumed (in order from top to bottom):

YYYY-MM-DD hh:mm:ss
YYYY/MM/DD hh:mm:ss
DD/MM/YY hh:mm:ss
DD/MM/YYYY hh:mm:ss
DD-MM-YY hh:mm:ss
DD-MM-YYYY hh:mm:ss
MMM DD, YYYY hh:mm:ss
hh:mm:ss
hh:mm A
the date/time format for the current locale
the time format for the current locale

Unspecified Date Elements

If the user does not provide a particular time unit when specifying a date, the following default values are used:

day, month	01
year	the current year
hour	the current hour
minute, second	00

If the hour, minute and second are all missing, the current time is used. If the year and month are both missing, the current year and month are used.

Edit Format and Display Format

The display format specified by XmNxrtFldDisplayFormatString is automatically added to the end of the list specified by XmNxrtFldEnterFormatList if it is not already present. This ensures that the values displayed in date fields are recognized as dates when edited.

The first format in the list specified by XmNxrtFldEnterFormatList is the format used to display the date while it is being edited. If XmNxrtFldEnterFormatList is not defined, and a display format is specified by XmNxrtFldDisplayFormatString, the display format is used as the edit format. (The XmNxrtFldEditFormat resource is not used by date fields.)

For details on XmNxrtFldDisplayFormatString, refer to section 3.8 on page 53.

3.4 Validation

XRT/field validates the value stored in a field whenever a keystroke action, paste action, user-exit action or programmatic change action occurs. In other words, validation occurs when the user types a keystroke, pastes text into the field or exits the field, and whenever the program changes the value. To see exactly where validation fits into the XRT/field editing process, see Figure 16, located on on page 28.

You can specify that one or more of the following validations are to be performed on a field:

- Maximum number of lines in field
- Maximum line length of field
- Valid and invalid characters for field
- Validation mask
- Data type checking
- Value range for field
- List of valid values (pick list) for field
- Validation callbacks

The validation methods are performed in the order listed above.

If a value is flagged as invalid, XRT/field performs error handling, described in section 3.5 on page 49.

Changing Validation Resources

If any of the resources that control the validation process are changed (such as XmNxrtFldMin), the field is automatically validated again, to ensure that the field value is still valid.

The value of XmNxrtFldInternalValue is unchanged even if the re-validated value is now invalid.

Obtaining the Validation Result

The XmNxrtFldIsValid resource indicates whether the current displayed value is valid.

Internal Values of Invalid Fields

If a field value is invalid, XmNvalue is copied to XmNxrtFldInternalValue whenever possible. If XmNvalue cannot be copied (if it is of the wrong data type), the default value of XmNxrtFldInternalValue is used. This default depends on the widget subclass:

- For date and string fields, this value is NULL.
- For integer fields, this value is XRTFLD_MAXLONG.
- For float and currency fields, this value is XRTFLD_MAXDOUBLE.

3.4.1 Number of Lines

If the field is a string field and the XmNeditMode resource is set to XmMULTI_LINE_EDIT, the user is allowed to store multiple lines of text in a field. To specify an upper limit on the number of lines of text, use the resource XmNxrtFldMaxLines.

By default, XmNxrtFldMaxLines is set to XRTFLD_MAXINT, which means that a field can contain any number of lines.

For more information on XmNeditMode, refer to section 3.11 on page 58.

3.4.2 Line Length

Use the XmNxrtFldMaxLineLength resource to specify the maximum length of each line in the field.

By default, XmNxrtFldMaxLineLength is XRTFLD_MAXINT, which means that a character string stored in a field can be of any length.

If an invalid field length is detected, the error callbacks specified by XmNxrtFldErrorCallback are only called if the XmNwordWrap resource is set to False. (This is its default value.)

The behavior of XmNxrtFldMaxLineLength differs from that of the XmText resource XmNmaxLength. The XmNmaxLength resource specifies the maximum number of characters, including newlines, that can appear in the *entire* XmText widget or a field, not just on an individual line. If XmNxrtFldMaxLineLength is set, XmNmaxLength is ignored.

3.4.3 Valid and Invalid Characters

You can ensure that a value stored in a field is considered valid only when it consists of specified characters. There are two ways to accomplish this:

- Use XmNxrtFldValidChars to list the characters that *can* appear in a valid value.
- Use XmNxrtFldInvalidChars to list the characters that *cannot* appear in a valid value.

For example, to ensure that only digits can appear in a particular field, set XmNxrtFldValidChars as follows:

```
XtVaSetValues(field,
    XmNxrtFldValidChars, "0123456789",
    NULL);
```

The following uses XmNxrtFldInvalidChars to invalidate values containing the # or % character:

```
XtVaSetValues(field,
    XmNxrtFldInvalidChars, "#%",
    NULL);
```

If both XmNxrtFldValidChars and XmNxrtFldInvalidChars are set, the list specified by XmNxrtFldInvalidChars is checked first.

By default, XmNxrtFldValidChars and XmNxrtFldInvalidChars are set to NULL, and the value is not checked for either valid or invalid characters.

3.4.4 Validation Masks

While valid and invalid character lists are useful, they do not indicate which characters are allowed to appear in a particular position of a field. To do this, specify a *validation mask* using the XmNxrtFldMask resource. Validation masks can only be used by string and date fields.

A validation mask is a text string. Each character in the validation mask corresponds to a character in the value of the field, and specifies which characters are valid at that field position.

The following characters can be included as part of a validation mask:

#	Any digit, minus sign, comma, decimal point, or plus sign is valid at this field position.
@	Any digit is valid at this field position.
H	Any hexadecimal digit (digits plus A-F) is valid at this field position.
U	Any alphabetic character is valid at this field position. Lower case characters are converted to upper case.
L	Any alphabetic character is valid at this field position. Upper case characters are converted to lower case.
A	Any alphabetic character is valid at this field position. No case conversion is performed.
*	Any character is valid at this field position.
?	Any character in the list specified by the XmNxrtFld-ValidChars resource is valid at this field position.
\\	The next character is to be treated as a literal, even if it is one of the characters listed above.

Mask validation is performed whenever the user types a character or exits the widget, and when the program changes the value in the field.

Here is an example of a statement that defines a mask:

```
XtVaSetValues(field,
    XmNxrtFldMask, "@@@*AAA",
    NULL);
```

In this example, the mask "@@@*AAA" specifies that the field must contain seven characters. The first three characters must be digits, the fourth can be any character, and the last three must be alphabetic.

Note: When using drag and drop or the paste action, if the pasted value is longer than the length of the mask, the text value is truncated.

Using Literals in Masks

Any character that appears in a mask and is not one of the special characters described above is treated as a *literal*. Literals are displayed in the field to help the user recognize what the field is expected to contain. They cannot be deleted by the user.

For example, if the mask "@@@-@@@@" is specified for a field, the dash is treated as a literal and is displayed in the field as shown in Figure 25.

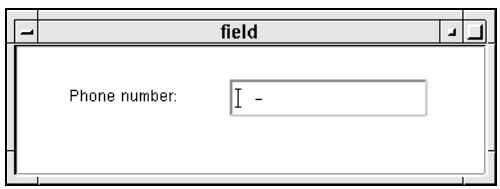


Figure 25 An empty field with a mask literal displayed

After the first three characters are typed and verified, the cursor jumps to the first character position following the dash, as shown in Figure 26:

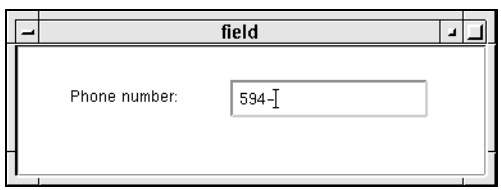


Figure 26 A partially filled field containing a mask literal

The user does not need to type the dash, and it cannot be erased.

If a proportional font is being used, literals displayed in the field are automatically shifted as the user enters characters.

Literals and the Stored Value

Although literals contained in masks are displayed by the field, they do not become part of the value stored in the `XmNxrtFldInternalValue` resource. Since the literals are not actually stored, you do not need to change your program to change how the mask is displayed – all you have to do is change the mask. For example, Figure 27 shows the phone number widget with two different area code masks.

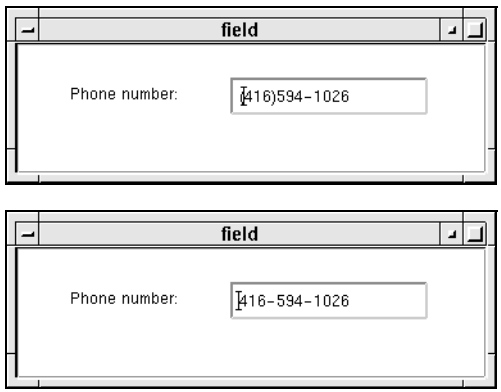


Figure 27 The phone number widget with different area code masks

In both cases, the value of `XmNxrtFldInternalValue` is the string "`4165941026`", which is the phone number with the literals removed.

To access the actual value stored in the field, including literals, get the value of the `XmNvalue` resource. In this example, `XmNvalue` contains "`(416)594-1026`" before the mask is altered, and contains "`416-594-1026`" afterwards.

Masks and Overstrike Mode

Overstrike mode is always set when a mask is specified by `XmNxrtFldMask`. The value of the resource `XmNxrtFldOverstrike` (discussed in section 4.14 on page 85) is ignored.

Matching a Partial Mask

By default, when a mask is defined, XRT/field treats a field value as valid only if:

- Every character in the field matches its corresponding character in the mask.
- The length of the value is identical to that of the mask.

If a value meets the first condition listed above but not the second, a *partial match* has occurred. To tell XRT/field that partial matches are to be treated as valid, use the `XmNxrtFldMatchMask` resource.

`XmNxrtFldMatchMask` can be set to any of the following values:

`XRTFLD_MATCH_PARTIAL_L_TO_R` accept partial matches; match mask starting from the left

XRTFLD_MATCH_PARTIAL_R_TO_L	accept partial matches; match mask starting from the right
XRTFLD_MATCH_ALL	accept complete matches only (the default)

Here is an example that matches phone numbers:

```
XtVaSetValues(field,
    XmNxrtFldMask,          "@@@-@@@@ ext @@@",
    XmNxrtFldMatchMask,     XRTFLD_MATCH_PARTIAL_L_TO_R,
    NULL);
```

In this case, permitting left-to-right partial matching allows you to match phone numbers for which an extension is not provided.

A mask is treated as partially matched if the unmatched portion of the value consists entirely of spaces:

```
XtVaSetValues(field,
    XmNxrtFldMask,          "@@@-@@@@ ext @@@",
    XmNxrtFldMatchMask,     XRTFLD_MATCH_PARTIAL_L_TO_R,
    XmNxrtFldInternalValue, "@@@@@@@    ",
    NULL);
```

Here, the phone number is matched and the extra spaces are ignored.

Specifying Visual Cues

One way to make fields using masks easier for users is to specify visual cues that appear when the field is first displayed. For example, if the user is required to enter a date in the "DD/MM/YY" format, you can specify a visual cue as follows:

```
XtVaSetValues(field,
    XmNvalue,       "DD/MM/YY",
    XmNxrtFldMask,  "@@/@@/@@",
    NULL);
```

Figure 28 shows what is displayed when this field appears.

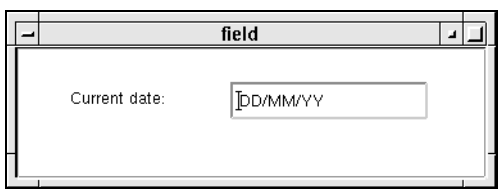


Figure 28 A date field containing a DD/MM/YY template

The XmNvalue resource contains the text string to be displayed in the field. This string, "DD/MM/YY", is not a valid value for the date field specified in this statement. Its sole purpose is to serve as a template indicating what should be typed where.

Since the template is not a valid date, the widget is displayed using the error background and foreground colors specified by the XmNxrtFldErrorBackground and XmNxrtFldErrorForeground resources.

Validating Pasted Values

When a mask is set for the target field, a pasted or dragged-and-dropped value is only valid if:

- the pasted value contains only valid characters defined by the mask.
- the pasted value contains literals at positions required by the mask and valid characters in all other positions.

In both cases, partial match left-to-right is considered valid and values longer than the mask length are truncated. Text can be dropped at any position in the field.

3.4.5 Data Type

Unless a mask has been specified by `XmNxrtFldMask`, XRT/field automatically performs datatype validation. For example, changing the contents of an integer field to `3.14` is flagged as invalid because `3.14` is not an integer.

A date stored in a date field is only valid if it is actually a legal date. For example, the date `31/04/95` is not treated as valid if the edit format `DD/MM/YY` is being used, since April 31st is not a legal date.

3.4.6 Value Range

Another way to validate a value is to specify upper and lower limits using the `XmNxrtFldMin` and `XmNxrtFldMax` resources.

For string fields, the values of `XmNxrtFldMin` and `XmNxrtFldMax` are text strings. A value assigned to the field is alphabetically compared with `XmNxrtFldMin` and `XmNxrtFldMax` using the *strcoll()* routine. For example, if `XmNxrtFldMin` is `alpha` and `XmNxrtFldMax` is `beta`, the value `bat` is valid, since `bat` is between `alpha` and "`beta`".

For date fields, `XmNxrtFldMin` and `XmNxrtFldMax` are pointers to structures of type *struct tm* (the ANSI time structure). A date is valid if it is later than `XmNxrtFldMin` and earlier than `XmNxrtFldMax`.

For other fields, a value is valid if it is greater than `XmNxrtFldMin` and smaller than `XmNxrtFldMax`.

If `XmNxrtFldMax` is set but `XmNxrtFldMin` is not, the value in a field has a maximum value but no minimum. Similarly, if `XmNxrtFldMin` is set but `XmNxrtFldMax` is not, the value has a minimum value but no maximum.

A validation range, if defined, is not checked after a keystroke event has occurred (since the user has not finished typing the value yet).

3.4.7 List of Valid Values

The `XmNxrtFldPickList` resource specifies a list of valid values for a field (a pick list). If this resource is defined and the resource `XmNxrtFldMatchPickList` is set to `True`, a value entered into a field is only valid if it matches an element of the list.

For string fields, the value of XmNxrtFldPickList is a NULL-terminated array of text strings:

```
String stringlist[] = {
    "valid1",
    "valid2",
    NULL };

XtVaSetValues(field,
    XmNxrtFldPickList,      stringlist,
    XmNxrtFldMatchPickList, True,
    NULL);
```

Here, the pick list consists of the strings valid1 and valid2.

When a pick list is specified for a field and the field is empty, the first element of the pick list becomes the initial value of the field. In the example above, the widget's initial value is valid1.

For integer fields, the value of XmNxrtFldPickList is an array of values of type *long*, with the first element of the array being the number of elements in the pick list:

```
long intlist[] = { 3,
    40,
    57,
    118 };

XtVaSetValues(field,
    XmNxrtFldPickList,      intlist,
    XmNxrtFldMatchPickList, True,
    NULL);
```

Here, the pick list consists of the integers 40, 57 and 118.

For float and currency fields, the value of XmNxrtFldPickList is an array of values of type *double*. The first value of the array is the number of elements in the pick list.

For date fields, the value of XmNxrtFldPickList is a NULL-terminated array of pointers to structures of type *struct tm* (the ANSI time structure).

Pick list validation checking is not performed after a keystroke event (since the user has not finished typing the value yet).

Pick Lists and Masks

If XmNxrtFldMask has been used to define a mask for a field, each element of the pick list must not contain mask literals. For example:

```
String phonenumbers[] = {
    "4165551212",
    "4165941026",
    NULL };
XtVaSetValues(field,
    XmNxrtFldPickList,      phonenumbers,
    XmNxrtFldMatchPickList, True,
    XmNxrtFldMask,          "(@@@)@@@-@@@@",
    NULL);
```

Here, the parentheses and dash in the mask are not included in the pick list elements.

3.4.8 Validation Callbacks

You can supply your own customized validation methods for a field. To do this, define validation callbacks using the XmNxrtFldValidateCallback resource.

Each validation callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;                          /* read-only */
    XEvent          *event;                          /* read-only */
    XrtFldAction    action;                          /* read-only */
    int             num_calls;                       /* read-only */
    XmTextPosition  startPos, endPos;                /* read-only */
    XmTextPosition  currInsert;                      /* read-only */
    XmTextPosition  newInsert;
    XmTextBlock     text;
    Boolean         doit;                       /* initially True */
    String          value;
} XrtFldValidateCallbackStruct;
```

The elements of this structure that are most commonly used are *action*, *text*, *value* and *doit*. (For descriptions of the other elements in this structure, refer to XmNxrtFldValidateCallback on page 115.)

action is the action that is currently being processed; it is one of XRTFLD_ACTION_KEYSTROKE (keystroke action), XRTFLD_ACTION_EXIT (user-exit action), XRTFLD_ACTION_PASTE (paste action) or XRTFLD_ACTION_SETVALUE (programmatic change action). See section 2.7 on page 24 for a description of these actions, and of the XRT/field editing process.

text contains the text to be validated.

value contains the entire value stored in the field (even if *text* does not). This value includes any conversions or completions previously specified.

doit indicates whether the value stored in *text* should be treated as valid. If your callback routine determines that the value is invalid, the routine must set *doit* to False.

The following callback routine checks whether a date in the form "MM/DD/YY" is in one of the first six months of the year. Later months are flagged as invalid:

```
void
validateCB(Widget widget, XtPointer client_data,
    XrtFldValidateCallbackStruct *cs)
{
    int value;

    if (cs->action == XRTFLD_ACTION_KEYSTROKE) return;
    if (cs->text->ptr == NULL) return;
    value = 10 * (cs->text->ptr[0] - '0') +
        (cs->text->ptr[1] - '0');
    if (value > 6) {
        cs->doit = False;
    }
}
```

Validation callbacks cannot set the XmNvalue and XmNxrtFldInternalValue resources. (Any attempts to do so will be ignored.) The value stored in the field can only be accessed from the XrtFldValidateCallbackStruct structure.

3.5 Handling Invalid Fields

Whenever validation of a field fails, the following occurs:

- XRT/field invokes any defined error callbacks.
- The field displays with the error foreground and background colors.

When an error is detected and the value of the field is displayed, it is displayed as is. The edit format specified by XmNxrtFldEditFormat and the display format specified by XmNxrtFldDisplayFormat are not used.

Error Callbacks

To specify error callbacks to be invoked whenever an invalid value is detected in a field that is losing focus, use the XmNxrtFldErrorCallback resource. This resource contains a standard XtCallbackList.

Each error callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;                          /* read-only */
    XEvent          *event;                          /* read-only */
    int             text_reason;                     /* read-only */
    XrtFldAction    action;                          /* read-only */
    XrtFldError     error;
    Boolean         is_modified;
    XmTextPosition  startPos, endPos;                /* read-only */
    XmTextPosition  currInsert;                      /* read-only */
    XmTextPosition  newInsert;
    XmTextBlock     text;
    XmString        error_string;                    /* read-only */
    Boolean         display;                  /* initially False */
    Boolean         traverse;                  /* initially True */
} XrtFldErrorCallbackStruct;
```

The most commonly used elements in this structure are *action*, *error*, *text* and *error_string*. (For descriptions of the other elements in this structure, see XmNxrtFldErrorCallback on page 115.)

action is the action that is currently being processed; it is one of XRTFLD_ACTION_KEYSTROKE (keystroke action), XRTFLD_ACTION_PASTE (paste action), XRTFLD_ACTION_EXIT (user-exit action) or XRTFLD_ACTION_SETVALUE (programmatic change action). See section 2.7 on page 24 for a description of these actions, and of the XRT/field editing process.

error indicates which validation operation detected the invalid value, and is one of the following:

XRTFLD_ERROR_EXCEED_LENGTH	the length of a line in the value is greater than `XmNxrtFldMaxLineLength`.
XRTFLD_ERROR_EXCEED_LINES	the number of lines in the value is greater than `XmNxrtFldMaxLines`.
XRTFLD_ERROR_IS_NULL	the field is empty and `XmNxrtFldAllowNull` is set to `False`.
XRTFLD_ERROR_INVALID_CHAR	a character in the value is either in the `XmNxrtFldInvalidChars` list, or is not in the `XmNxrtFldValidChars` list.
XRTFLD_ERROR_INVALID_DATATYPE	a character in the value is not appropriate for the data type of the field (for example, an alphabetic character in an integer field).
XRTFLD_ERROR_NOMATCH_MASK	the value does not match the mask specified by `XmNxrtFldMask`.
XRTFLD_ERROR_NOMATCH_PICKLIST	the value is not part of the pick list specified by `XmNxrtFldPickList`.
XRTFLD_ERROR_OUT_OF_RANGE	the value is either larger than the upper limit specified by `XmNxrtFldMax`, or is smaller than the lower limit specified by `XmNxrtFldMin`.
XRTFLD_ERROR_VALIDATECB	the value has been flagged as invalid by a validation callback. (The validation callback does this by setting its *doit* structure member to `False`.)

To override any error other than `XRTFLD_ERROR_INVALID_DATATYPE`, set *error* to `XRTFLD_ERROR_NONE`. This tells XRT/field to treat the value as valid.

error_string is the message to be displayed by the application when an error occurs, and is the value of the `XmNxrtFldErrorString` resource. This message is of type XmString (compound string). The following error callback displays an error message in an XmLabel widget:

```
void
errorCB(Widget widget, XtPointer client_data,
    XrtFldErrorCallbackStruct *cs)
{
    XtVaSetValues(error_label,
        XmNlabelString, cs->error_string,
        NULL);
}
```

Error Colors

For information on error colors and XRT/field, refer to section 3.10 on page 57.

3.6 Conversion

XRT/field allows you to convert values stored in fields from one form to another. You can specify conversion whenever a user-exit action, keystroke action, paste action or programmatic change action occurs.

Conversion operations, when defined, are performed before validation takes place. (See Figure 16 on on page 28 for information on where conversion fits into the editing process.)

The following conversion operations can be specified using resources:

- Case conversion can be specified with `XmNxrtFldCase`. When set to `XRTFLD_CASE_UPPER`, the value stored in the field is converted to upper case. When set to `XRTFLD_CASE_LOWER`, the value is converted to lower case. By default, `XmNxrtFldCase` is set to `XRTFLD_CASE_AS_IS`, and no case conversion is performed.
- Case conversion of individual characters in a string can be specified as part of a validation mask. For more information on masks and validation, refer to section 3.4.4 on page 42. (Mask case conversions override conversions specified with `XmNxrtFldCase`.)

All other conversions are specified using conversion callbacks.

Conversion Callbacks

To register a conversion callback, add it to the list specified by `XmNxrtFldConvertCallback`. This resource contains a standard XtCallbackList.

Each conversion callback routine is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason            reason;                 /* read-only */
    XEvent                  *event;                 /* read-only */
    XrtFldAction            action;                 /* read-only */
    XmTextPosition          startPos, endPos;       /* read-only */
    XmTextPosition          currInsert;             /* read-only */
    XmTextPosition          newInsert;
    XmTextBlock             text;
    String                  value;
} XrtFldConvertCallbackStruct;
```

action is the action that is currently being processed; it is one of `XRTFLD_ACTION_KEYSTROKE` (keystroke action), `XRTFLD_ACTION_EXIT` (user-exit action), `XRTFLD_ACTION_PASTE` (paste action) or `XRTFLD_ACTION_SETVALUE` (programmatic change action).

text contains the text to be converted. Its contents depend on the value of *action*. If *action* is `XRTFLD_ACTION_KEYSTROKE`, *text* contains the last character typed by the user. If *action* is `XRTFLD_ACTION_PASTE`, *text* contains the text produced by the last paste or drag and drop operation. If *action* is `XRTFLD_ACTION_EXIT` or `XRTFLD_ACTION_SETVALUE`, *text* contains the entire contents of the field.

startPos contains the number of characters between the start of the field and the first character in *text*. *endPos* contains the number of characters between the start of the

field and the last character in *text*. If *text* contains a single character, *startPos* and *endPos* are equivalent.

currInsert specifies the current position of the cursor. *newInsert* specifies the desired new position of the cursor. (By default, this is the same as *currInsert*.)

The *value* member contains the entire value. Its format depends on *action*. If the conversion callback is triggered by a user-exit action or a keystroke action, *value* is provided in the field's edit format (specified by `XmNxrtFldEditFormat` if it is defined for that subclass). In the case of a paste action, *value* is displayed in edit format or display format, depending on the keyboard focus. If the conversion callback is triggered by a programmatic change action, *value* is provided in the field's display format (specified by `XmNxrtFldDisplayFormat` if it is defined for that subclass).

The following is an example of a conversion callback for a string field. Each keystroke typed by the user is examined. If the keystroke is the first character of the value, it is converted to upper case. Otherwise, it is converted to lowercase.

```
void
convertCB(Widget widget, XtPointer client_data,
    XrtFldConvertCallbackStruct *cs)
{
    if (cs->action != XRTFLD_ACTION_KEYSTROKE) return;
    if (cs->text->ptr == NULL) return;
    if (cs->currInsert == 0) {
        cs->text->ptr[0] = toupper(cs->text->ptr[0]);
    } else {
        cs->text->ptr[0] = tolower(cs->text->ptr[0]);
    }
}
```

Callback routines *cannot* set the `XmNvalue` or `XmNxrtFldInternalValue` resources. (Any attempts to do so will be ignored.) The only way to access or change the field value from a callback is by manipulating the *text* or *value* member of the callback structure. (If both *text* and *value* are modified, *value* is used.) To change the field value, free the storage used by the old value, allocate storage for the new value, and assign the new storage to the pointer specified by *text* or *value*.

3.7 Completion

When the user exits a widget, you can add to any incomplete value stored in a field. This completion step takes place after the conversions described in the previous section have been performed, and before validation is performed. (See Figure 16 on on page 28 for where completion fits into the editing process.)

To complete a value, specify completion callbacks by setting the `XmNxrtFldCompleteCallback` resource. (This resource contains a standard

XtCallbackList.) Each completion callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    String          value;
} XrtFldCompleteCallbackStruct;
```

value is the new value of the field; completion is performed by modifying this value. The value can also be changed by freeing the value and allocating and filling a new one. *value* is provided in the field's edit format. (Edit formats are discussed in section 3.2 on page 36.)

The following is an example of a completion callback. This callback adds the ".c" suffix to a file name if no suffix is provided.

```
void
completeCB(Widget widget, XtPointer client_data,
    XrtFldCompleteCallbackStruct *cbs)
{
    String value;

    if (cbs->value && !strstr(cbs->value, ".c")) {
        value = XtMalloc(strlen(cbs->value) + 3);
        strcpy(value, cbs->value);
        strcat(value, ".c");
        XtFree(cbs->value);
        cbs->value = value;
    }
}
```

Since the length of the value is changed by this callback, it frees the string containing the old value and allocates a string for the new value.

Note that completion callbacks cannot set the `XmNvalue` and `XmNxrtFldInternalValue` resources. (Any attempts to do so will be ignored.) The value stored in the field can only be accessed from the `XrtFldCompleteCallbackStruct` structure. To change this value, free the string pointed to by *value*, create a string containing the new value, and set *value* to point to it.

3.8 Display Format Specification

Integer, float and currency fields allow you to specify how the field value displays when it is not being edited by setting the `XmNxrtFldDisplayFormat` resource.

`XmNxrtFldDisplayFormat` has the same possible values as `XmNxrtFldEditFormat`. For information on these values, see the description of the edit format in section 3.2 on page 36.

The display format specified by `XmNxrtFldDisplayFormat` is only used when the value is valid. If the value is invalid, the value is displayed as the user typed it.

If `XmNxrtFldDisplayFormat` is not specified or is defined to be `XRTFLD_FORMAT_DEFAULT`, the value is displayed in the format used when editing.

The following specifies one format for editing and another for display:

```
XtVaSetValues(field,
    XmNxrtFldInternalValue,     255,
    XmNxrtFldEditFormat,        XRTFLD_FORMAT_GENERAL,
    XmNxrtFldDisplayFormat,     XRTFLD_FORMAT_OCTAL,
    NULL);
```

Here, the value stored in the widget is displayed in octal notation, but is edited in general (decimal) notation. When the user begins editing the field, its value is converted to decimal. Before it is displayed, it is automatically converted to octal.

When a value is displayed in octal notation, the prefix `0o` is displayed to indicate that the number is octal:

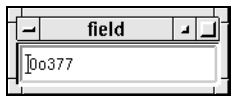


Figure 29 A value displayed in octal notation

Similarly, the prefix `0b` indicates binary notation, and `0x` indicates hexadecimal notation.

Custom Display Formats

When displaying a value in an integer, float or currency field, you can define a *sprintf* format string that specifies exactly how the value is to be displayed when not being edited. To do this, set `XmNxrtFldDisplayFormat` to `XRTFLD_FORMAT_GENERAL`, and set the resource `XmNxrtFldDisplayFormatString` to the *sprintf* string you want to use.

For example, to display an integer value in a right-justified 10-character field:

```
XtVaSetValues(field,
    XmNxrtFldDisplayFormat, XRTFLD_FORMAT_GENERAL,
    XmNxrtFldDisplayFormatString, "%10ld",
    NULL);
```

The *sprintf* string used by `XmNxrtFldDisplayFormatString` must contain an integer format specifier (such as `%d` or `%o`) if the widget is an integer field. Float and currency fields must use a floating-point format specifier (such as `%e`, `%f` or `%g`). Any flag, field-width, precision or size conversion specification can be provided with the format specifier. The *sprintf* string can contain literals.

Date Field Formats

The `XmNxrtFldDisplayFormatString` resource controls the display of date fields. In this case, this resource contains a character string consisting of some combination of the following:

D	numeric day of month (1-31)
DD	numeric day of month with leading zero
W	abbreviated weekday name
WW	weekday name
J	Julian date (day number of year) (1-366)
JJ	Julian date with leading zero
M	numeric month of year
MM	numeric month with leading zero
MMM	abbreviated month name
MMMM	month name
Y, YY	year within current century (00-99)
YYYY	year, including century
H	hour (0-12)
HH	0-12 hour with leading zero
h	hour (0-23)
hh	0-23 hour with leading zero
m	minute (0-59)
mm	minute with leading zero
s	second (0-59)
ss	second with leading zero
A or P	AM/PM representation
Z	time zone abbreviation
\\	next character is a literal

All other characters are treated as literals, and displayed without processing.

Note that the time zone always defaults to the time zone of the local machine. You cannot alter the time zone via an XrtDateField widget.

If this resource is not specified, the first item in the list stored in `XmNxrtFldEnterFormatList` is used.

3.9 Exit Callbacks

When the user is about to exit a field whose value has been validated, you can add post-validation processing by specifying exit callbacks. These callbacks are referenced via the XmNxrtFldExitCallback resource, which contains a standard XtCallbackList.

Each exit callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason        reason;                     /* read-only */
    XEvent              *event;                     /* read-only */
    int                 text_reason;                /* read-only */
    Boolean             is_modified;
    String              value;                      /* read-only */
    Boolean             traverse;               /* initially True */
} XrtFldExitCallbackStruct;
```

value is the new value of the field (the value to be displayed). This value is in the format specified by XmNxrtFldDisplayFormat.

text_reason indicates which XmText callback triggered this exit callback, and is one of the following:

XmCR_LOSING_FOCUS triggered by XmNlosingFocusCallback

XmCR_ACTIVATE triggered by XmNactivateCallback

text_reason is only set to XmCR_ACTIVATE when the user attempts to exit the field by pressing the Return key. All other exit methods set *text_reason* to XmCR_LOSING_FOCUS. (For more information on the relationship between XmText callbacks and field callbacks, refer to section 4.15 on page 85.)

is_modified contains the current value of XmNxrtFldIsModified, which indicates whether the value was last changed by the user or the program. Changing *is_modified* changes the value of this resource.

traverse indicates whether the user should be allowed to traverse out of the field when he or she presses the Return or Tab key. By default, *traverse* is True, and traversal takes place.

traverse and *text_reason*, together, allow you to disable traversal when the user presses Return:

```
if (cs->text_reason == XmCR_ACTIVATE) {
    cs->traverse = False;
}
```

This lets your field widget simulate the behavior of an XmText widget (which doesn't allow traversal when the user presses Return).

Exit callbacks are often not the best way to do post-validation processing, as they don't allow you to examine fields as a group. To examine several fields at once, use the *XrtFldCheckFields()* procedure. For information on *XrtFldCheckFields()*, see section 4.11 on page 81.

3.10 Field Colors

Background

A field's background color is determined by the `XmNbackground` resource. Change the value of this resource to change the background color.

Foreground

A field's foreground color is determined by the `XmNforeground` resource. Change the value of this resource to change the foreground color.

Error Colors

When an invalid value is detected, the field's background and foreground colors are set to the values indicated by the `XmNxrtFldErrorBackground` and `XmNxrtFldErrorForeground` colors. By default, these colors are red and white, respectively. (On monochrome monitors, these default colors are black and white.) To leave the field colors unchanged when an error occurs, set `XmNxrtFldErrorBackground` and `XmNxrtFldErrorForeground` to `XRTFLD_UNCHANGED_COLOR`.

Colors for Default Values

XRT/field allows you to define different background and foreground colors for fields containing their default values. To specify the colors for widgets containing the default value, set the resources `XmNxrtFldDefaultBackground` and `XmNxrtFldDefaultForeground`. This lets a user determine at a glance which fields have not yet been set.

If `XmNxrtFldDefaultBackground` and `XmNxrtFldDefaultForeground` are not set, widgets containing default values are displayed using the normal field background and foreground colors.

For more information on default values for widgets, refer to section 4.1 on page 59.

The Pixel Data Type

All color resources recognized by fields have values of type Pixel.

To convert a color name (such as "red") to a value of type Pixel, convert it using `XtVaTypedArg`. For example, to specify an error background color of green:

```
XtVaSetValues(field,
    XtVaTypedArg, XmNxrtFldErrorBackground,
    XmRString, "green", strlen("green")+1,
    NULL);
```

For more information on `XtVaTypedArg` or the Pixel data type, refer to your Motif documentation.

3.11 Specifying Multi-Line Text

By default, fields handle a single line of text. If a newline character is entered by the user, the newline and any characters following it are ignored.

String fields allow you to let users input multiple lines of text. To do this, set the `XmNeditMode` resource to `XmMULTI_LINE_EDIT`. To revert to a single line of text, set `XmNeditMode` to its default value of `XmSINGLE_LINE_EDIT`.

Note: Motif 1.2 programs must set `XmNrows` to a value greater than 1 when using multiline text fields. Because of a bug in Motif 1.2, widgets that support multiple lines but only display one line do not behave properly.

4

Advanced XRT/field Programming

Default Field Value ■ *Enter Prompts and Enter Callbacks* ■ *Using XRT/field with XRT/table*
Auto Matching Completion ■ *Drop-Down List Creation* ■ *XrtComboBox Programming*
XrtSpinBox Programming ■ *XrtCalendar Programming* ■ *Date Editor Popup*
Binding Actions to Keys ■ *Checking Multiple Fields* ■ *Date Conversion*
Controlling Scrolling ■ *Defining Mask Characters* ■ *XrtField and XmText Callbacks*
Using C++ with XRT/field ■ *Using UIL with XRT/field* ■ *Converting Resource Values to Strings*
Internationalization

This chapter covers topics that programmers of advanced XRT/field applications will find useful. It assumes that the reader is already familiar with XRT/field.

4.1 Default Field Value

XRT/field allows you to assign a *default value* for a field, displayed whenever the field is empty.

String fields are "empty" when they contain `NULL` or the empty string. Other widget subclasses are "empty" when they contain `NULL` or whitespace characters (blanks, tabs and newline characters).

The `XmNxrtFldDefaultValue` resource specifies a field's default value. The data type of this resource is the same as the data type of the `XmNxrtFldInternalValue` resource.

The following code creates an integer field and sets its default value to 42:

```
field = XtVaCreateManagedWidget("field",
    xmXrtIntFieldWidgetClass,   toplevel,
    XmNxrtFldDefaultValue,      42,
    NULL);
```

If XmNxrtFldDefaultValue is not defined, the following default values are used:

Integer fields	0
Float and currency fields	0
String fields	NULL
Date fields	the widget creation time

Note: If you set the field to empty and it has a default value, the field is set to that default value on update.

Displaying No Default Value

To display no default value when the field has not been set, set XmNxrtFldDefaultValue as follows:

- For integer fields, set XmNxrtFldDefaultValue to XRTFLD_NOVALUE.
- For float and currency fields, set XmNxrtFldDefaultValue to XRTFLD_NOVALUE_FLOAT.
- For string and date fields, set XmNxrtFldDefaultValue to NULL.

Default values, like values set by the user, are converted, completed, and checked for validity. If a default value is valid, XmNvalue and XmNxrtFldInternalValue are set to this value.

Checking for Default Values

Use the XmNxrtFldIsDefault resource to distinguish between default values and assigned values.

If XmNxrtFldIsDefault is True, the field was empty, and the value stored in the field is the default value. If XmNxrtFldIsDefault is False, the value of the field is the result of a previous assignment. (This resource is read-only – it cannot be set by the program.)

Note that XmNxrtFldIsDefault is False even when the value assigned by the user or program happens to be identical to the value specified by XmNxrtFldDefaultValue.

Default Value Colors

XmNxrtFldDefaultBackground and XmNxrtFldDefaultForeground allow you to set the background and foreground colors for fields containing default values.

By default, XmNxrtFldDefaultBackground and XmNxrtFldDefaultForeground are set to XRTFLD_UNCHANGED_COLOR, which means that fields containing default values are displayed in the normal field display colors (specified by XmNbackground and XmNforeground).

For more information on colors in XRT/field, refer to section 3.10 on page 57.

Allowing Empty Fields

Because some applications cannot deal with empty fields, XRT/field allows you to specify whether or not an empty field is valid. To disallow an empty field, set XmNxrtFldAllowNull to False.

By default, XmNxrtFldAllowNull is True, and empty fields are permitted.

4.2 Enter Prompts and Enter Callbacks

XRT/field allows you to specify a message to be displayed by the application when a field gains keyboard focus. This message is known as the *enter prompt* for the field.

To define an enter prompt, set the XmNxrtFldPromptString resource to the text string to use as the prompt. (This string must be of type XmString, which is the Motif compound string type.) Then, define an enter callback and add it to the list specified by the XmNxrtFldEnterCallback resource.

Each enter callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;                         /* read-only */
    XEvent          *event;                         /* read-only */
    String          value;                          /* read-only */
    XmString        prompt_string;                  /* read-only */
} XrtFldEnterCallbackStruct;
```

prompt_string is the current value of the XmNxrtFldPromptString resource. *value* is the value that will be displayed when the user edits the field. This is the contents of XmNxrtFldInternalValue converted to edit format.

Here is an example of a code fragment that uses XmNxrtFldPromptString to define an enter prompt, and then writes the prompt string into the message area of the application's MainWindow widget:

```
string = XmStringCreate("Enter your login userid",
    XmFONTLIST_DEFAULT_TAG);
field = XtVaCreateManagedWidget("field",
    xmXrtStringFieldWidgetClass,          toplevel,
    XmNxrtFldPromptString,                string,
    NULL);

void
enterCB(Widget widget, XtPointer client_data,
    XrtFldEnterCallbackStruct *cs)
{
    XtVaSetValues(MessageArea, XmNlabelString,
        cs->prompt_string, NULL);
}
```

4.3 Using XRT/field with XRT/table

By default, an XmText widget is used to edit and validate text. XRT/table allows you to use XRT/field's validation, editing and display capabilities. To do this, perform the following four steps:

1. Ensure that the XRT/field library (libxrtfield.a) appears before the XRT/table library (libxrttable.a) in the list of libraries you specify when linking your compiled program.

2. Call the *XrtTblEnableXrtField()* procedure before creating any tables:

```
XrtTblEnableXrtField();
...
table = XtVaCreateManagedWidget("table",
    xtXrtTableWidgetClass, toplevel,
    NULL);
```

3. Use the `XmNxrtTblDataTypeContext` resource to specify the type of XRT/field widget you want to use. The following types are available:

`XRTTBL_TYPE_FIELD_CURRENCY`	currency field
`XRTTBL_TYPE_FIELD_DATE`	date/time field
`XRTTBL_TYPE_FIELD_FLOAT`	floating-point field
`XRTTBL_TYPE_FIELD_INT`	integer field
`XRTTBL_TYPE_FIELD_NONE`	
`XRTTBL_TYPE_FIELD_STRING`	string field
`XRTTBL_TYPE_FIELD_COMBO_CURRENCY`	currency combo box
`XRTTBL_TYPE_FIELD_COMBO_DATE`	date combo box
`XRTTBL_TYPE_FIELD_COMBO_FLOAT`	floating-point combo box
`XRTTBL_TYPE_FIELD_COMBO_INT`	integer combo box
`XRTTBL_TYPE_FIELD_COMBO_STRING`	string combo box
`XRTTBL_TYPE_FIELD_SPIN_CURRENCY`	currency spin box
`XRTTBL_TYPE_FIELD_SPIN_DATE`	date spin box
`XRTTBL_TYPE_FIELD_SPIN_FLOAT`	floating-point spin box
`XRTTBL_TYPE_FIELD_SPIN_INT`	integer spin box
`XRTTBL_TYPE_FIELD_SPIN_STRING`	string spin box

An XRT/field widget of the appropriate type is automatically created when needed. Only one of each type of field is created, and is reused as the user traverses.

4. To set resources for your XRT/field widget, use the `XmNxrtTblFieldArgListContext` resource. Its value must be a static list of Args, and the last element of the list must be `NULL`:

```
static Arg args[3];

XtSetArg(args[0], XmNxrtFldMask, "###-####");
XtSetArg(args[1], XmNxrtFldAllowNull, False);
XtSetArg(args[2], NULL, NULL);
XtVaSetValues(table,
    XmNxrtTblContext, XrtTblSetContext(5, 5),
    XmNxrtTblFieldArgListContext, args,
    NULL);
```

When editing is completed, the edited value is stored in the table's `XrtTblCellValues` structure.

The *table_entry* demo, located in *$XRTHOME/src/common_demos/table_entry*, provides an example of a program that uses XRT/field widgets with XRT/table.

Note: It is up to the user to manage the memory for the `XmNXrtTblFieldArgList`, as it is not copied internally.

Fields in Resource Files

The name of a created XRT/field widget reflects its type:

Widget Type	Field Name	Child Field Name	Child Arrow Name	Child Arrow Name
XRTTBL_TYPE_FIELD_CURRENCY	"currency_field"			
XRTTBL_TYPE_FIELD_DATE	"date_field"			
XRTTBL_TYPE_FIELD_FLOAT	"float_field"			
XRTTBL_TYPE_FIELD_INT	"int_field"			
XRTTBL_TYPE_FIELD_STRING	"string_field"			
XRTTBL_TYPE_FIELD_COMBO_CURRENCY	"combo_currency_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_DATE	"combo_date_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_FLOAT	"combo_float_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_INT	"combo_int_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_COMBO_STRING	"combo_string_field"	"field"	"arrow"	
XRTTBL_TYPE_FIELD_SPIN_CURRENCY	"spin_currency_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_DATE	"spin_date_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_FLOAT	"spin_float_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_INT	"spin_int_field"	"field"	"decr_arrow"	"incr_arrow"
XRTTBL_TYPE_FIELD_SPIN_STRING	"spin_string_field"	"field"	"decr_arrow"	"incr_arrow"

Resources for this field can be set in a resource file.

Retrieving a Field

To retrieve an XRT/field widget of a particular type, call the *XrtTblGetXrtFieldByType()* method:

```
field = XrtTblGetXrtFieldByType(table,
        XRTTBL_TYPE_FIELD_CURRENCY);
```

If a field of the appropriate type does not exist, it is created (unmanaged).

Once you have retrieved a field, you can use it to format or update a cell value. For example, the following code converts an integer value to hexadecimal format and puts it in a cell:

```
field = XrtTblGetXrtFieldByType(table,
        XRTTBL_TYPE_FIELD_INT);
XtVaSetValues(field,
    XmNxrtFldDisplayFormat, XRTFLD_FORMAT_HEX,
    XmNxrtFldInternalValue, 53,
    NULL);
value = XrtFldGetDisplayValue(field);
XtVaSetValues(table,
    XmNxrtTblContext,XrtTblSetContext(4, 3),
    XmNxrtTblCellValueContext,value,
    NULL);
XtFree(value);
```

The XRT/field method *XrtFldGetDisplayValue()* retrieves the string that the field would display. This string is then stored in the cell by setting the `XmNxrtTblCellValueContext` resource.

You cannot set the following XRT/field resources through a resource file or through a call to *XtSetValues()* on the field:

```
XmNxrtFldMask
XmNxrtFldPickList
XmNxrtFldMatchPickList
XmNxrtFldPickListIndex
```

To set these resources, you need to use `XmNxrtTblFieldArgListContext`, since these resources are reset internally by the table as the user moves from cell to cell in the table. To have them take affect in the new cell, they have to be reloaded from `XmNxrtFldArgListContext`, with the exception of `XmNxrtFldPickList`. The table uses its own picklist resources (such as `XmNxrtTblStringPickListContext`) and when a user moves into a new cell, the table's own picklist resources are applied to the field. If no XRT/table picklist is provided for the cell, then the picklist from the `XmNxrtFldArgListContext` is used, if present. If the picklist exists in both the table and the `XmNxrtTblFieldArgListContext`, the picklist in the table takes precedence.

It is strongly recommended that you let the table manage the picklists for you (using the various `XmNxrtTbl...PickList` resources), rather than using `XmNxrtTblFieldArgListContext` to specify picklists. If you decide to use the latter method, you will also have to allocate, maintain and deallocate the memory required for these picklists. XRT/table will not handle it for you.

Further XRT/table Information

For more information on XRT/table and its features, refer to the *XRT/table Programmer's Guide and Reference Manual.*

4.4 Auto Matching Completion

The `XmNxrtFldAutoComplete` resource allows auto matching if a pick list has been set. For example, if a picklist contained the values *red*, *green,* and *blue*, then entering

"g" while XmNxrtFldAutoComplete is True jumps to the value *green*, as soon as the user types it in. The characters typed in the field by the user appear in normal foreground and background colors, but the auto-completed portion of the value from the pick list displays in reverse video. If the partial value entered is not a member of the pick list, XmNxrtFldDefaultValue replaces the invalid value.

Note that if XmNxrtFldMatchPickList is set to False, a value not present in the pick list is allowed and is not replaced with XmNxrtFldDefaultValue.

4.5 Drop-Down List Creation

Normally, a drop-down menu or an XrtCalendar popup is not created until the first time it is displayed. However, there are times when users want to specify properties for popups when the field widget is created. XmNxrtFldCreatePopups controls whether popups are created immediately or on the first use. If XmNxrtFldCreatePopups is True, the popups are created immediately. Setting this resource to False once a popup has been created does *not* destroy the popup.

The widget ID of the drop-down menu is accessed through XmNxrtFldMenuList, while the widget ID of the XrtCalendar widget is accessed through XmNxrtFldCalendar.

4.6 XrtComboBox Programming

The following sections describe customizing the behavior of the XrtComboBox widget. For an introduction to this widget, refer to section 1.5 on page 13.

Default Field Settings

When an XrtComboBox is created, its child field has the following resources set by default:

```
XmNcursorPositionVisible    False
XmNeditable                 False
XmNeditMode                 XmSINGLE_LINE_EDIT
XmNmarginHeight             2
```

These resources cannot be set from resource files, but can be set programmatically:

```
XtVaSetValues(combo,
    XmNeditable,               True,
    XmNcursorPositionVisible,  True,
NULL);
```

The popup menu created when the user presses the arrow button has the following XmList resources set by default:

```
XmNautomaticSelection       False
XmNborderWidth              0
```

XmNscrollBarDisplayPolicy	XmAS_NEEDED
XmNselectionPolicy	XmBROWSE_SELECT
XmNselectedItems	NULL
XmNselectedItemCount	0
XmNspacing	0

These resources cannot be set by the application. Other XmList resources can be set from resource files. For example, the following changes the background color for popup menus:

```
combo.*.picklist.background: green
```

Specifying the Arrow Class

The XmNxrtComboArrowClass resource allows you to specify the widget class of the arrow button. By default, this widget class is xmArrowButtonWidgetClass.

XmNxrtComboArrowClass can only be set at the time the XrtComboBox is created.

The position of the arrow button relative to the field is determined by the value of XmNstringDirection.

Retrieving the Child Widget IDs

You can obtain the widget ID of either of the children of the XrtComboBox widget. To obtain the widget ID of the field, retrieve the value of the XmNxrtComboField resource. Similarly, to obtain the widget ID of the arrow widget, retrieve the value of the XmNxrtComboArrow resource.

The XmNxrtComboField and XmNxrtComboArrow resources cannot be set.

Specifying the Pick List Search Policy

When a pick list is displayed, the end-user can search for an item in the pick list by pressing one or more keys. The XmNxrtFldSearchPolicy resource controls this popup menu search behavior, and can be set to either XRTFLD_SEARCH_SINGLE (each keypress resets the search, which is the default) or XRTFLD_SEARCH_MULTICHAR (each keypress refines the search).

Figure 30 provides examples that illustrate the behavior of XmNxrtFldSearchPolicy.

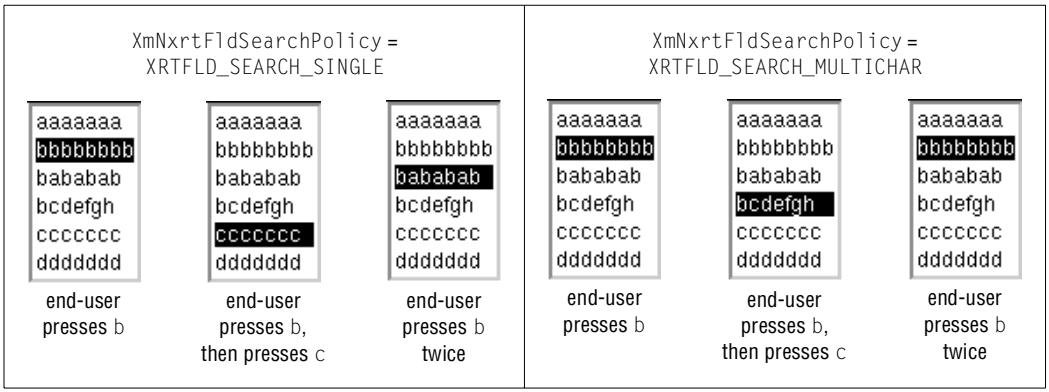


Figure 30 Popup menu search policies

Modifying the Pick List

To obtain the widget ID of the pick list, get the value of the `XmNxrtFldMenuList` resource. This resource is read-only; it cannot be set.

The pick list is an XmList widget; once you have its widget ID, you can set any resources not controlled by the parent Combo Box widget. The following resources are controlled by the parent widget and cannot be set programmatically: `XmNborderWidth`, `XmNheight`, `XmNitemCount`, `XmNitems`, `XmNselectionPolicy`, `XmNvisibleItemCount`, and `XmNwidth`.

The following example specifies the font in which the pick list items are to be displayed by setting the pick list's `XmNfontList` resource:

```
static String fontlist =
"-adobe-helvetica-medium-r-normal--12-120-75-75-p-67-iso8859-1,\
-adobe-helvetica-bold-o-normal--14-140-75-75-p-82-iso8859\
-1=highlight";
XtVaGetValues(combobox,
    XmNxrtFldMenuList, &menulist,
    NULL);
XtVaSetValues(menulist,
    XtVaTypedArg, XmNfontList, XmRString,
    fontlist, strlen(fontlist)+1,
    NULL);
```

Note that if `XmNxrtFldSearchPolicy` is set to `XRTFLD_SEARCH_MULTICHAR`, the highlight font is used to display keys entered by the end-user when searching in a pick list, as shown in Figure 31. (For more information on `XmNxrtFldSearchPolicy`, see Appendix A on page 93 .)

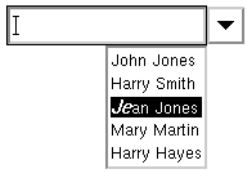


Figure 31 Using a highlight font in a pick list

For more information on Combo Box user interaction, see section 2.3 on page 21.

Modifying the Text Field

When a Combo Box widget is created, a child Text widget is also automatically created; this Text widget contains the currently selected item.

The following Text resources can be changed or retrieved by passing the combo box widget ID to *XtVaSetValues()* or *XtVaGetValues()*:

`XmNautoShowCursorPosition`	`XmNblinkRate`
`XmNborderWidth`	`XmNcolumns`
`XmNcursorPosition`	`XmNcursorPositionVisible`
`XmNeditable`	`XmNeditMode`
`XmNfontList`	`XmNhighlightThickness`
`XmNmarginHeight`	`XmNmarginWidth`
`XmNmaxLength`	`XmNpendingDelete`
`XmNresizeWidth`	`XmNrows`
`XmNselectThreshold`	`XmNselectionArray`
`XmNselectionArrayCount`	`XmNsensitive`
`XmNsource`	`XmNshadowThickness`
`XmNtopCharacter`	`XmNvalue`
`XmNverifyBell`	`XmNwordWrap`

To set other text resources or add callbacks, you must retrieve the widget ID of the text field. To do this, get the value of the `XmNxrtComboField` resource. This resource is read-only; it cannot be set.

Menu Alignment

The `XmNxrtFldMenuAlignment` resource enables you to specify the alignment of the popup menu relative to the text field of the Combo Box. Valid values for this resource, and their effects, are shown in Figure 32.

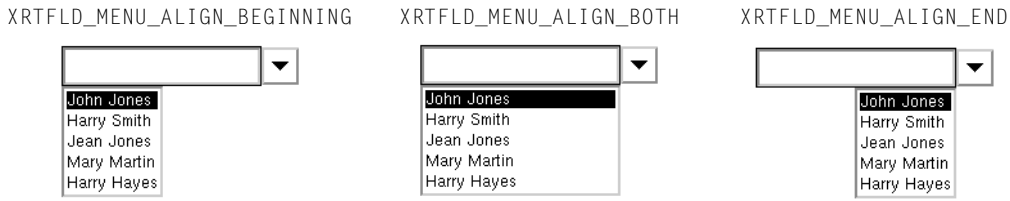


Figure 32 Combo Box menu alignment

By default, `XmNxrtFldMenuAlignment` is set to `XRTFLD_MENU_ALIGN_BOTH`.

Specifying Child Widget Spacing

By default, there is no space between the field and the arrow in an XrtComboBox. To put space between these two child widgets, set the `XmNxrtComboSpacing` resource, as shown in Figure 33.

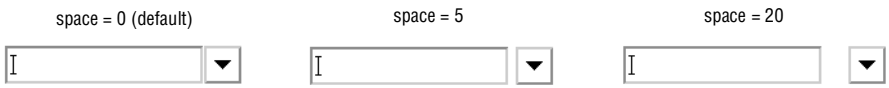


Figure 33 Space between the field and the arrow

Here, space is measured in pixels.

Specifying the Arrow Type

To specify the type of arrow to be displayed, se the `XmNxrtComboArrowType` resource. This resource can only be set at the time the widget is created. Valid values are shown in Figure 34.

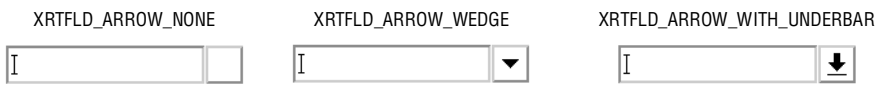


Figure 34 XrtComboBox arrow types

The default value is `XRTFLD_ARROW_WEDGE`.

The position of the arrow button relative to the text field is determined by the value of `XmNstringDirection`.

In addition to the above arrow types, you can also set `XmNxrtComboArrowType` to `XRTFLD_ARROW_CUSTOM`. This indicates that you are supplying your own arrow type.

4.7 XrtSpinBox Programming

The following sections describe customizing the behavior of the XrtSpinBox widget. For an introduction to this widget, refer to section 1.5 on page 13.

Default Field Settings

When an XrtSpinBox is created, its child field has the following resources set by default:

`XmNcursorPositionVisible`	`False`
`XmNeditable`	`False`
`XmNeditMode`	`XmSINGLE_LINE_EDIT`

These resources cannot be set from resource files, but can be set programmatically.

Adding Additional Children

You can add other widgets, including additional fields, to an XrtSpinBox widget. To do this, use the standard Motif *XtVaCreateManagedWidget()* procedure, specifying the XrtSpinBox as the parent widget.

If a spin box contains more than one child field widget, the arrow buttons act on the field that currently has the input focus.

Specifying the Arrow Location

The `XmNxrtSpinArrowLayout` resource allows you to specify the position of the arrow buttons in an XrtSpinBox. This resource can be set to any of the following values:

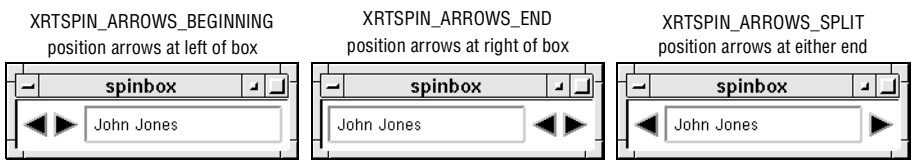


Figure 35 Possible values for XmNxrtSpinArrowLayout

By default, `XmNxrtSpinArrowLayout` is set to `XRTSPIN_ARROWS_SPLIT`.

If a parent widget defines the resource `XmNstringDirection`, and it is set to `XmSTRING_DIRECTION_R_TO_L`, `XRTSPIN_ARROWS_BEGINNING` positions arrows at the right of the box, and `XRTSPIN_ARROWS_END` positions arrows at the left of the box. (If `XmNxrtSpinArrowLayout` is set to `XRTSPIN_ARROWS_SPLIT`, changing `XmNstringDirection` has no effect on the location of the arrows.)

Specifying the Arrow Orientation

The `XmNxrtSpinArrowOrientation` resource allows you to specify the orientation of the arrows in an XrtSpinBox. This resource can be set to the following:

`XmHORIZONTAL`	use left and right arrows
`XmVERTICAL`	use up and down arrows

If XmNxrtSpinArrowLayout is set to XRTSPIN_ARROWS_BEGINNING or XRTSPIN_ARROWS_END and XmNxrtSpinArrowOrientation is set to Xm_VERTICAL, the vertical arrows appear one atop the other:

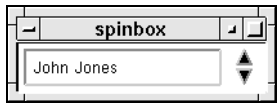


Figure 36 Stacked vertical arrows in an XrtSpinBox

Specifying the Spin Data Type

The XmNxrtSpinType resource indicates the type of data displayed in an XrtSpinBox. It can be set to either of the following values:

XRTSPIN_TYPE_PICKLIST cycle the items specified by XmNxrtFldPickList

XRTSPIN_TYPE_COUNTER spin range defined by XmNxrtFldMin and XmNxrtFldMax, with increment specified by XmNxrtFldCounterIncrement

By default, XmNxrtSpinType is set to XRTSPIN_TYPE_PICKLIST.

Specifying the Spin Policy

The resource XmNxrtFldSpinPolicy allows you to control the type of counter spinning used when an XrtDateField is within a spin box. XmNxrtFldSpinType must be set to XRTSPIN_TYPE_COUNTER for this to apply. XmNxrtFldSpinPolicy can have three values: XRTFLD_SPIN_FIELD, XRTFLD_SPIN_SUBFIELD, and XRTFLD_SPIN_WRAP.

When the spin policy is XRTFLD_SPIN_FIELD, the field uses XmNxrtFldCounterIncrement to spin. The spin policy XRTFLD_SPIN_WRAP is similar to XRTFLD_SPIN_FIELD, except that it allows the spin values to wrap. (i.e. going one past the last value will display the first value, and vice versa.)

A spin policy of XRTFLD_SPIN_SUBFIELD means that the date subfield where the cursor currently resides is spun. Spinning subfields only applies to XrtDate fields. This resource is ignored for all other types.

When spinning subfields, only the first entry in the XmNxrtFldEnterFormatList is used. All other ones are ignored.

Specifying an Arrow Class

You can specify the widget class for either or both of the child arrow widgets. To specify the widget class of the increment arrow (the arrow pointing either up or right, depending on XmNxrtSpinArrowLayout), set the value of the XmNxrtSpinIncrArrowClass resource when creating the XrtSpinBox. Similarly, to specify the widget class of the decrement arrow (the arrow pointing either down or left), set the value of the XmNxrtSpinDecrArrowClass resource.

The XmNxrtSpinIncrArrowClass and XmNxrtSpinDecrArrowClass resources can only be set at the time the XrtSpinBox is created. By default, the widget class for both arrows is xmArrowButtonWidgetClass.

Retrieving the Child Widget IDs

You can obtain the widget ID of any of the pre-defined children of the XrtSpinBox widget.

- To obtain the widget ID of the first field, retrieve the value of the `XmNxrtSpinField` resource.
- To obtain the widget ID of the increment arrow button, retrieve the value of the `XmNxrtSpinIncrArrow` resource.
- To obtain the widget ID of the decrement arrow button, retrieve the value of the `XmNxrtSpinDecrArrow` resource.

None of these resources can be set.

Specifying Child Widget Spacing

By default, there is no space between the field and the arrow in an XrtSpinBox. To put space between all child widgets, set the `XmNxrtSpinSpacing` resource. Space is measured in pixels; the default is 0 (no space).

Specifying the Arrow Type

To specify the type of arrow to be displayed, se the `XmNxrtSpinArrowType` resource. This resource can only be set at the time the widget is created. Valid values are the same as those for the XrtComboBox's `XmNxrtComboArrowType` resource. The default value is `XRTFLD_ARROW_WEDGE`.

4.8 XrtCalendar Programming

The following sections describe customizing the behavior of the XrtCalendar widget. For an introduction to this widget, refer to section 1.6 on page 17.

All typical XrtSpinBox functions and settings can be used with the XrtCalendar's spin box elements. See "XrtSpinBox Programming" on page 70.

On any action in the calendar, including spinning the month, time or year fields, `XmNxrtModifyCalendarCallback` is called.

Daygrid Header

The symbols for the days of the week are displayed in the header and are dependant on the locale. The font, background and foreground of the header can be set. The user can also set the number of header characters allowed with `XmNxrtCalendarHeaderNumChars`. This is *1*, by default, which displays the weekdays as *SMTWTFS*. Setting it to *2* displays *SuMoTuWeThFrSa* and so on. As expected, setting this value greater than or equal to the longest name or `MAX_INT` causes the full names to be used. The resource `XmNxrtCalendarFullDayNames` controls whether *full name* means the actual full name of the day or the locale's abbreviated version of the day name.

The first day of the week is normally Sunday (Day 0), but this can be changed by setting `XmNxrtCalendarFirstDayOfWeek`.

Calendar Appearance

The calendar widget tries to become as large as needed, based on `XmNfontList` and `XmNxrtCalendarHeaderFontList`, but if it can't, the calendar is centered in the available area. Note there are no scrollbars to accommodate oversized calendar displays.

The default colors are set using `XmNforeground` and `XmNbackground`, but the user can also set the header foreground and background, the selected foreground and background and the foreground and background for special days.

There are four margin resources (top, bottom, left, right) which specify the margin around the entire calendar.

A calendar's cell highlight thickness can be set with `XmNxrtCalendarCellHighlightThickness`. The highlight applies to the current cell, not the whole calendar. It is the rectangle displayed around the current cell when the daygrid gets the focus.

The user can also set the margin for each individual cell on the daygrid, not just for the calendar as a whole.

Special Days

The user can have certain days, such as holidays or recurring deadlines, stand out from the other days on the daygrid by denoting them as *special days*. The user does this by using `XmNxrtCalendarSpecialDays`, which points to a list of special days for the widget. Each entry in the list specifies a special day or list of special days for the widget.

For example, if type is `XRTCALENDAR_SPECIAL_DAY_BY_DATE`, then *day*, *month* and *year* fields determine a special day. If type is `XRTCALENDAR_SPECIAL_DAY_BY_DAY_OF_WEEK`, then *day_of_week*, *week*, *month* and *year* fields determine a special day. If any of the fields are set to `XRTCALENDAR_ALL`, then the `XrtCalendarSpecialDay` is interpreted for each valid value of that field. For example, to mark one special day every year, set the type to `XRTCALENDAR_SPECIAL_DAY_BY_DATE`, set *day* and *month* fields to the appropriate value, and set *year* to `XRTCALENDAR_ALL`.

The following code sample is from the *calendar.c* example included with XRT/field. It uses Special Days to make Christmas Day and New Year's Day stand out from normal days.

```
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/XrtField.h>

static char *fallbacks[] = {
"*.XmXrtCalendar.background: white",
"*.XmXrtCalendar.background: grey",
"*.XmXrtCalendar.foreground: black",
"*.XmXrtCalendar.xrtCalendarSpecialDayBackground: yellow",
"*.XmXrtCalendar.xrtCalendarSpecialDayForeground: red",
NULL
};

/* Set Christmas and New Years Day as special days */
static XrtCalendarSpecialDay days[] = {
        { XRTCALENDAR_SPECIAL_DAY_BY_DATE, 25, 0, 0, 11, XRTCALENDAR_ALL},
        { XRTCALENDAR_SPECIAL_DAY_BY_DATE, 1, 0, 0, 0, XRTCALENDAR_ALL},
        };
static XrtCalendarSpecialDay *spec_days[] = {
        &days[0],
        &days[1],
        NULL
        };

{
XtAppContext app_context;
Widget calendar, toplevel;

/* Initialize Intrinsics */
toplevel = XtVaAppInitialize(&app_context, "toplevel", NULL, 0,
&argc, argv, fallbacks, NULL);

calendar = XtVaCreateManagedWidget("calendar",
        xmXrtCalendarWidgetClass, toplevel,
        XmNxrtCalendarSpecialDays, spec_days,
        XmNxrtCalendarCreateTime, TRUE,
        NULL);


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

Figure 37 Code sample showing special day usage

These special days are made to stand out from the rest of the days on the daygrid by using `XmNxrtCalendarSpecialDayForeground` and `XmNxrtCalendarSpecialDayBackground` to alter the foreground and/or background color of the denoted days. If either of these resources is not set, the unset resource defaults to `XmNforeground` or `XmNbackground`, respectively.

Special Days can also be set using a resource file. The following resource file entry sets Christmas Day and New Year's Day to be Special Days:

```
*.XmXrtCalendar.xrtCalendarSpecialDays: (SPECIAL_DAY_BY_DATE 25 0 0 11
XRTCALENDAR_ALL) (SPECIAL_DAY_BY_DATE 1 0 0 0 XRTCALENDAR_ALL)
```

4.9 Date Editor Popup

The `XrtDateField` and `XrtComboBox` widgets allow an `XrtCalendar` widget to be used as their popup. When the user holds down the CTRL key and then presses the down arrow key while the cursor is in a field widget, or when a combo box's arrow button is clicked, the calendar widget pops up. `XRTFLD_POPUP_PICKLIST` is the default for all types of fields, except XrtDateField. `XRTFLD_POPUP_CALENDAR` is the default for XrtDateField. Users can change this to a pick list by setting `XmNxrtFldPopupType`. This can only be set on widget create.

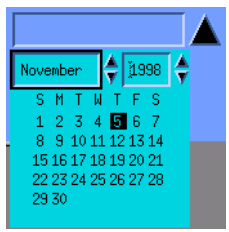


Figure 38 Calendar widget popped-up from a combo box.

The date editor tries to adhere to the `XmNxrtFldMenuAlignment` combo box resource; however, if the calendar is bigger than the field and the alignment is `XRTFLD_MENU_ALIGN_BOTH`, part of the calendar widget may be truncated.

Similar to the "Modifying the Pick List" discussion earlier in this chapter, the `XmNxrtFldCalendar` resource points to the widget ID of the `XrtCalendar` widget. Thus, since the resources are controlled by the parent widget `XrtCalendar`, `XmNwidth` cannot be modified. Calendar resources `XmNxrtCalendarCreateMonth`, `XmNxrtCalendarCreateTime`, `XmNxrtCalendarCreateYear` can only be set from a resource file. All of the other resources are modifiable and can also be initialized with a resource file. The following line in a resource file would set the background of a combo box's calendar popup to hot pink:

```
combo.*.datepopup.background:hotpink
```

The user can provide code to be executed when the popup appears or is dismissed by using Calendar callbacks. These function similarly to the menu callbacks. To use a Calendar callback, add callbacks to the list specified by `XmNxrtFldCalendarCallback`.

4.10 Binding Actions to Keys

XRT/field allows the following field actions to be bound to keys:

- *Calendar()*, which displays a popup date editor.
- *Cancel()*, which replaces the value currently being typed by the user with the last complete value entered.
- *Complete()*, which uses the valid list to complete the current value.
- *Menu()*, which displays a popup menu of the valid list items.
- *NoAction()*, which does nothing (effectively disabling the translation).
- *Overstrike()*, which sets or disables overstrike mode.
- *Scrollbars()*, which toggles the display of scrollbars.
- *Spin()*, which either displays the next item in the valid list or increments the current item.

For example, the following binds the *Complete()* action to the Return key:

```
String translations = "Key<Return>: Complete()";

XtOverrideTranslations(field,
    XtParseTranslationTable(translations));
```

The following sections describe these actions in more detail. For more information on binding actions to keys, refer to your Motif documentation.

4.10.1 The Calendar Action

The *Calendar()* action routine controls the display of a popup date editor calendar. The popup calendar opens to the default of today's date. From there, the user can change the date displayed in the XrtDateField.

Calendar takes one argument that specifies the display action:

`Display`	Display the date popup.
`Hide`	Hide the date popup.

Holding down the CTRL key and pressing the down-arrow toggles the calendar popup display for the XrtDateField.

Calendar Callbacks

XRT/field lets you provide code to be executed either when the popup calendar is displayed or just before it is dismissed. To do this, add callbacks to the list specified by `XmNxrtFldCalendarCallback`. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    struct tm       *date;
    Boolean         doit;       /* initially True */
} XrtFldCalendarCallbackStruct;
```

reason is either XRTFLD_REASON_CALENDAR_BEGIN (before the calendar popup has been displayed) or XRTFLD_REASON_CALENDAR_END (after the calendar popup has been dismissed).

date is a pointer to a time structure which stores the date and time selected. If the popup does not include the time spinbox, then the time fields will be filled in with zeros. *date* can only be changed if *reason* is XRTFLD_REASON_CALENDAR_END.

The behaviour of *doit* depends on the value of *reason*. If *reason* is XRTFLD_REASON_CALENDAR_END and *doit* is set to False, the selected date is disallowed.

4.10.2 The Cancel Action

The *Cancel()* action deletes the value currently being typed and replaces it with the value of the XmNxrtFldInternalValue resource, which contains the last complete value entered into the field.

Cancel() lets you add an "undo" feature to your applications.

4.10.3 The Complete Action

When the *Complete()* action is specified, XRT/field examines each element of the pick list specified by the XmNxrtFldPickList resource. If the value currently stored in the field matches part of an element in the pick list, the matched element is used to complete the value in the field.

For example, suppose the pick list consists of the values

```
January
April
July
October
```

and the value in the field is Apr. In this case, when *Complete()* is called, April becomes the value stored in the field.

If a pick list has been specified, and the value stored in the field does not match any of its elements, the terminal beeps and the value stored in the field remains unchanged.

Completion Callbacks

When the *Complete()* action is specified, the callbacks specified by the XmNxrtFldCompleteCallback resource are invoked. For a description of the structure passed to these callbacks, refer to section 3.7 on page 52.

Note that *Complete()* calls the completion callbacks even if XmNxrtFldPickList is defined. This gives you the freedom to complete a value using any rules you like.

4.10.4 The Menu Action

The *Menu()* action routine controls the display of a popup menu listing the pick list specified by the `XmNxrtFldPickList` resource. If the value stored in the field matches one of the elements in the pick list, the element is selected from the popup menu and is scrolled to.

Menu takes one argument that specifies the display action:

`Display` The popup menu containing the pick list is displayed. Each element of the pick list is displayed in the field's edit format.

`Hide` The popup menu is removed from view.

Here is a simple example of code that uses a popup menu:

```
String picklist[] = {"Monday", "Tuesday", "Wednesday",
    NULL};

int
main(int argc, char *argv[])
{
    Widget toplevel, field;

toplevel = XtInitialize(argv[0], "Example", NULL, 0,
    &argc, argv);
field = XtVaCreateManagedWidget("field",
        xmXrtStringFieldWidgetClass, toplevel,
        XmNxrtFldPickList,  picklist,
        NULL);
XtRealizeWidget(toplevel);
XtMainLoop();
}
```

Figure 39 shows the popup menu displayed when the Control and down-arrow keys are pressed.

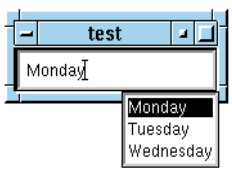


Figure 39 A popup menu displayed by a Menu() action

When the popup menu is created, the number of visible items is specified by the `XmNxrtFldVisibleItemCount` resource. A scrollbar is displayed if necessary.

Menu Callbacks

XRT/field lets you provide code to be executed either when the popup menu appears or just before it is dismissed. To do this, add callbacks to the list specified by

XmNxrtFldMenuCallback . Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    int             position;   /* read-only */
    String          value;
    Boolean         doit;       /* initially True */
} XrtFldMenuCallbackStruct;
```

reason is either XRTFLD_REASON_MENU_BEGIN (before the menu has been displayed) or XRTFLD_REASON_MENU_END (just before the menu is dismissed).

position is the index of the selected item in the pick list (starting from 1). If the user has not selected an item, *position* is set to XRTFLD_NOVALUE. *value* is the value that will be displayed in the field. *value* can only be changed if *reason* is XRTFLD_REASON_MENU_END.

The behavior of *doit* depends on the value of *reason*. If *reason* is XRTFLD_REASON_MENU_BEGIN and *doit* is set to False, the menu is not displayed (in this case, the application may perform a custom operation). If reason is XRTFLD_REASON_MENU_END and *doit* is set to False, the picked value is disallowed.

Pick Lists and Masks

If the XmNxrtFldMask resource defines a mask for a field, the elements of the menu displayed by *Menu()* are displayed in the format specified by the mask.

4.10.5 The Overstrike Action

The *Overstrike()* action routine controls the behavior of the XmNxrtFldOverstrike resource. Its effect depends on its argument:

- If *Overstrike()* is passed True, XmNxrtFldOverstrike is set to True.
- If *Overstrike()* is passed False, XmNxrtFldOverstrike is set to False.
- If *Overstrike()* is passed Toggle, XmNxrtFldOverstrike is set to the opposite of its current value.

The value of XmNxrtFldOverstrike is ignored if XmNxrtFldMask is set. See section 3.4.4 on page 42 for a description of field masks and validation.

4.10.6 The Scrollbars Action

The *Scrollbars()* action toggles scrollbar display by changing the values of the XmNxrtFldDisplayHorizScrollBar and XmNxrtFldDisplayVertScrollBar resources.

The first call to *Scrollbars()* turns scrollbar display off by setting both XmNxrtFldDisplayHorizScrollBar and XmNxrtFldDisplayVertScrollBar to XRTFLD_DISPSB_NEVER.

The second call to *Scrollbars()* turns scrollbar display on by setting XmNxrtFldDisplayHorizScrollBar to XRTFLD_DISPSB_ALWAYS, and setting XmNxrtFldDisplayVertScrollBar to XRTFLD_DISPSB_AS_NEEDED.

Subsequent calls to *Scrollbars()* toggle scrollbar display off and on, as described above.

4.10.7 The Spin Action

The *Spin()* action routine is designed for use with an XrtSpinBox widget. It displays either the next or the previous displayable value, depending on its argument:

- If *Spin()* is passed `Next`, the next displayable value is stored in the field.
- If *Spin()* is passed `Prev`, the previous displayable value is stored in the field.

When the user presses the right arrow button in an XrtSpinBox, *Spin()* is called with `Next` as its argument. When the user presses the left arrow button, *Spin()* is called with `Prev` as its argument.

The behavior of *Spin()* depends on the `XmNxrtSpinType` resource:

- If `XmNxrtSpinType` is set to `XRTSPIN_TYPE_PICKLIST`, *Spin()* cycles through the values specified by the `XmNxrtFldPickList` resource. When *Spin()* is called for the first time, the first value in the list specified by `XmNxrtFldPickList` is stored in the child field. Subsequent *Spin* operations select another element from the pick list and store it in the field. Wraparound is always performed when necessary (i.e., the first element in the list is assumed to be "following" the last element, and the last element "precedes" the first one).
- If `XmNxrtSpinType` is set to `XRTSPIN_TYPE_COUNTER`, *Spin()* cycles through the range of values bounded by the `XmNxrtFldMin` and `XmNxrtFldMax` resources. Each time *Spin()* is called, the value of `XmNxrtFldCounterIncrement` is added to, or subtracted from, the value stored in the field.

By default, `XmNxrtSpinType` is set to `XRTSPIN_TYPE_PICKLIST`.

Spin Callbacks

XRT/field lets you provide code which is executed whenever a *Spin()* action is performed. To do this, add callbacks to the list specified by `XmNxrtFldSpinCallback`. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;      /* read-only */
    XEvent          *event;      /* read-only */
    XrtFldDirection direction;   /* read-only */
    Boolean         at_boundary; /* read-only */
    Boolean         doit;       /* initially True */
} XrtFldSpinCallbackStruct;
```

reason is `XRTFLD_REASON_SPIN_BEGIN` or `XRTFLD_REASON_SPIN_END`. *direction* is `XRTFLD_DIRECTION_PREVIOUS` or `XRTFLD_DIRECTION_NEXT`. *at_boundary* is `True` if the `XmNxrtFldSpinType` resource is set to `XRTFLD_SPINTYPE_PICKLIST` and the current value is at the beginning or the end of the pick list. *doit* should be set to `False` if *reason* is `XRTFLD_REASON_SPIN_BEGIN` and cycling is not supposed to occur.

The Pick List Index

The XmNxrtPickListIndex resource indicates the position of the currently selected element in the pick list.

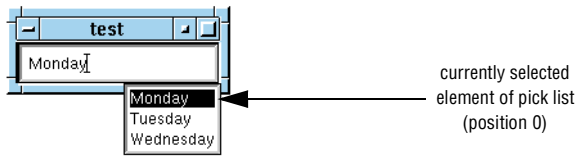


Figure 40 The value of XmNxrtPickListIndex

If the first element of the list is the current element, XmNxrtPickListIndex has the value 0. If the second element is the current element, XmNxrtPickListIndex has the value 1, and so on.

You can select a different item by setting XmNxrtPickListIndex. Setting its value to a value greater than the number of items in the list is equivalent to selecting the last item in the list.

If no item in the pick list is currently being displayed, XmNxrtPickListIndex has the value XRTFLD_NOVALUE.

4.11 Checking Multiple Fields

Sometimes, it is not possible to determine whether the value in one field is valid until another field is filled. (This happens, for example, when several fields are part of a form being filled out by the user.)

To perform multi-field validation, do the following:

- Create the fields you need as descendants of a manager widget.
- Once the user has finished entering data, examine all the child widgets at once by calling *XrtFldCheckFields()*.

XrtFldCheckFields() is defined as follows:

```
Boolean
XrtFldCheckFields(
    Widget          parent,
    XrtFldFieldData *data
    )
```

parent is the manager widget. *data* is a returned list of the fields that are descendants of the manager widget. This list is returned as a pointer to an *XrtFldFieldData* structure, which is defined as follows:

```
typedef struct {
    Widget *field_list;
    int num_field_list;
    Widget *invalid_list;
    int num_invalid_list;
    Widget *default_list;
    int num_default_list;
} XrtFldFieldData;
```

field_list lists all the fields that are descendants of the manager widget. *num_field_list* indicates how many of these widgets there are.

For convenience, the widgets in *field_list* whose values are invalid are listed in a separate list, stored in *invalid_list*. *num_invalid_list* indicates how many widgets with invalid values exist. Similarly, the widgets in *field_list* containing their default values are listed in *default_list*, and the number of widgets in this list is stored in *num_default_list*. (For an explanation of default values, refer to section 4.1 on page 59.)

Once you have the list returned by *XrtFldCheckFields()*, you can examine individual or pairs of fields in any combination you like, handling the cases where one widget is dependent on another. For example, the following push button callback (`XmNactivateCallback`) displays an error box if any field has not been typed in by the user:

```
Widget form, errorbox;

void
pushCB(Widget widget, XtPointer client_data,
    XmPushButtonCallbackStruct *cs)
{
    XrtFldFieldData data;

    XrtFldCheckFields(form, &data);
    if (data.num_default_list > 0) {
        XtManageChild(errorbox);
    } else {
    /*
     * here is where the form processing goes
     */
    }
    XrtFldFieldDataFree(&data);
}
```

When the callback is finished with the lists of widgets returned by *XrtFldCheckFields()*, it must call *XrtFldFieldDataFree()* to free the memory used to hold the list.

4.12 Date Conversion

Values supplied to date fields must be of type *struct tm* (an ANSI time structure). The *XrtFldCvtStringToTime()* procedure provides a convenient way of converting dates to *struct tm* structures.

For example, to set the value of a date field to be the current time, pass the string "NOW" to *XrtFldCvtStringToTime()*, as in the following example:

```
struct tm currtime;

XrtFldCvtStringToTime("NOW", NULL, &currtime);
XtVaSetValues(field,
    XmNxrtFldInternalValue, &currtime,
    NULL);
```

You can also convert a specified time to a *struct tm* using *XrtFldCvtStringToTime()*:

```
struct tm currtime;

XrtFldCvtStringToTime("Nov 11 1918 11:00",
    "MMM DD YYYY hh:mm", &currtime);
XtVaSetValues(field,
    XmNxrtFldInternalValue, &currtime,
    NULL);
```

This creates a date field and sets its date to November 11, 1918, 11:00 am. In this case, *XrtFldCvtStringToTime()* is passed the date and time to be converted, along with the format specifier to use. This format specifier is identical to those used by the XmNxrtFldEnterFormatList resource; see section 3.3 on page 38 for more details on possible format specifiers.

Setting XmNxrtFldCounterIncrement Using tm struct

XrtFldCvtStringToTime() can not only be used to set a date field, but can also be used in other interesting ways. For example, it can be used conjunction with a spinbox and increment counter:

```
/*
** Get the current time and set up the tm structure to increment the
** minutes by one
*/

XrtFldCvtStringToTime("NOW", "hh:mm:ss", &currtime);
XrtFldCvtStringToTime("00:01:00", "hh:mm:ss", &incrementtime);

spinbox = XtVaCreateWidget("spinbox",
            xmXrtSpinBoxWidgetClass,     rc,
            XmNxrtSpinType,              XRTSPIN_TYPE_COUNTER,
            XmNxrtSpinFieldType,         XRTFLD_FIELDTYPE_DATE,
            XmNxrtFldEnterFormatList,    formatList,
            XmNxrtFldInternalValue,      &currtime,
            NULL);

XtVaGetValues(spinbox, XmNxrtSpinField, &field, NULL);

XtVaSetValues(field,
  XmNxrtFldCounterIncrement, &incrementtime,
  NULL);
```

Here *XrtFldCvtStringToTime()* is called twice: once to populate a *struct tm* with the current time and again to populate a s*truct tm* for the increment. The current time structure is passed to the spinbox's internal value. A one minute increment, as represented by the incrementtime *tm struct* is passed to the counter increment resource.

Other Conversion Procedures

XRT/field provides procedures that convert a long integer to a *struct tm* and vice versa. The *XrtFldCvtLongToTime()* procedure converts a long integer to a *struct tm*, and the *XrtFldCvtTimeToLong()* procedure converts the *struct tm* back into a long integer.

These procedures are only useful if the date stored in the *struct tm* can fit into a long integer. (Any date roughly between 1902 and 2033 will fit.)

For more information on date conversion procedures, refer to Appendix B on page 129.

4.13 Controlling Scrolling

One of the useful features of fields is their built-in scrollbar capability, which is shown in Figure 41.

MOUSE: Use the scrollbars to scroll through the field

KEYBOARD: Press osfPageDown to scroll down a page
Press osfPageUp to scroll up a page

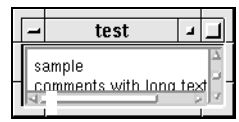


Figure 41 Scrolling behavior of fields

The `XmNxrtFldDisplayHorizScrollBar` and `XmNxrtFldDisplayVertScrollBar` resources control when to display horizontal and vertical scrollbars. Each of these resources can be set to any of the following values:

`XRTFLD_DISPSB_ALWAYS`	always display scrollbars
`XRTFLD_DISPSB_NEVER`	never display scrollbars
`XRTFLD_DISPSB_AS_NEEDED`	display scrollbars as needed

The default value for `XmNxrtFldDisplayHorizScrollBar` is `XRTFLD_DISPSB_AS_NEEDED`. The default for `XmNxrtFldDisplayVertScrollBar` is `XRTFLD_DISPSB_NEVER`.

These resources can also be controlled using the *Scrollbars()* action routine, which is described in section 4.10.6 on page 79.

Programming the Scrollbars

The scrollbars created by XRT/field are XmScrollBar widgets. The vertical scrollbar is named "`VertScrollBar`", and the horizontal scrollbar is named "`HorizScrollBar`". Vertical and horizontal scrollbars are automatically created when needed, and are destroyed when no longer needed.

The easiest way to change the attributes of these XmScrollBar widgets is using resource files. For example, to set the increment value for the horizontal scrollbar for a widget named "Myfield", add the following entry to your resource file:

```
*.Myfield.HorizScrollBar.increment: 3
```

4.14 Defining Mask Characters

To replace any or all of the special characters with characters of your own choosing, set the XmNxrtFldMaskChars resource. By default, this resource has the value "#@HULA*?\\", with each character in the string corresponding to a special character in masks:

Character	Value in Mask
#	The number (digit, minus sign, comma or decimal point) special character
@	The digit special character
H	The hexadecimal digit special character
U	The upper case alphabetic special character
L	The lowercase alphabetic special character
A	The alphabetic special character
*	The generic special character (which permits any character to appear)
?	The special character indicating a character in XmNxrtFldValidChars
\\	The escape character

To specify a special character of your own, replace a special character in the string with the character of your choice. For example, to use "!" as the digit special character, replace the "@" in the default string "#@HULA*?\\" with "!":

```
XtVaSetValues(field,
    XmNxrtFldMaskChars, "#!HULA?\\",
    NULL);
```

The string assigned to XmNxrtFldMaskChars is "#!HULA?\\", not "#!HULA?\". In the C and C++ programming languages, the backslash character \ is an escape character in strings; to include a backslash in a string, two backslash characters are required.

4.15 XrtField and XmText Callbacks

Because XrtField is a subclass of the XmText widget, XRT/field makes use of XmText callbacks internally. Many XmText callbacks trigger XRT/field callbacks.

The following XmText callbacks are used by fields:

- `XmNfocusCallback`, which is invoked whenever the field gains focus.
- `XmNmodifyVerifyCallback`, which is invoked whenever a programmatic change action or keystroke action occurs (before the value of the widget is changed).
- `XmNmotionVerifyCallback`, which is invoked when literals in a mask are skipped over.
- `XmNvalueChangedCallback`, which is invoked after the value of the widget is changed.
- `XmNactivateCallback`, which is invoked when the user presses the Return key.
- `XmNlosingFocusCallback`, which is invoked when the user tabs to another field or moves to another window.

When one of these XmText callback lists is invoked, the registered field callbacks are invoked. The following table lists the field callbacks associated with particular XmText callbacks:

	XmNactivateCallback	XmNfocusCallback	XmNlosingFocusCallback	XmNmodifyVerifyCallback	XmNvalueChangedCallback
XmNxrtFldCompleteCallback	●		●	●	
XmNxrtFldConvertCallback	●		●	●	●
XmNxrtFldEnterCallback		●			
XmNxrtFldErrorCallback	●		●	●	
XmNxrtFldExitCallback	●		●		
XmNxrtFldValidateCallback	●		●	●	●

Each XmText callback listed here triggers the field callbacks shown. Note that the `XmNxrtFldErrorCallback` list is only triggered if an error has occurred. Similarly, the `XmNxrtFldExitCallback` list is only triggered if an error has not occurred.

Retrieving the Callback Reason

When an `XmNxrtFldErrorCallback` or `XmNxrtFldExitCallback` procedure is invoked, you can determine which of the XmText callbacks triggered it. This allows you to, for example, define processing that only takes place when the user presses the Return key. For more details on error callbacks, refer to section 3.5 on page 49; for information on exit callbacks, refer to section 3.9 on page 56.

Specifying XmText Callbacks

If you specify your own XmText callbacks, they are added to the callback list after the field callbacks already specified. For example, if you specify an `XmNfocusCallback` for your widget, this callback is invoked after the callbacks specified by `XmNxrtFldEnterCallback` are invoked.

Normally, you do not need to specify XmText callbacks for fields, as the field callbacks are sufficient.

4.16 Using C++ with XRT/field

XRT/field provides a C++ interface to its widgets. To use it, include the header file *$XRTHOME/include/Xm/XrtField.hxx* in your C++ program. This file defines an XrtField class with the following methods:

- A constructor that calls *XmCreateXrtField()*.
- A destructor that calls *XtDestroyWidget()*.
- A method for each of the C procedures that takes an XRT/field widget as its first argument. For example, the method for *XrtFldGetField()* is *getField()*.
- A method to set each settable resource. For example, the method to set `XmNxrtFldEditFormat` is *setEditFormat()*.
- A method to retrieve each resource. For example, the method to retrieve `XmNxrtFldEditFormat` is *getEditFormat()*.
- Methods to perform common Xt procedures: *addCallback()*, *destroy()*, *getDisplay()*, *getName()*, *getParent()*, *isManaged()*, *isRealized()*, *manageChild()*.

4.17 Using UIL with XRT/field

XRT/field allows you to use OSF's User Interface Language (UIL) to create widgets. Since UIL only supports datatypes used by OSF's Motif widgets, some XRT/field resources cannot be set using UIL, but can be set in an application with an `MrmNcreateCallback`. See Appendix H on page 167 for a complete list of UIL arguments and their datatypes.

To create a widget with UIL:

- Include the XRT/field UIL header file *XrtField.uil* in your application;
- Call *XrtFldMrmInitialize()* immediately after calling *MrmInitialize()*.

4.18 Converting Resource Values to Strings

Resource converters convert resource values from one format to another. This allows you to convert values stored in resource files from strings to the format that the widget understands (such as Booleans and integers). XRT/field automatically

registers its built-in resource converters to convert from strings and all of the XRT/field-specific resource types.

XRT/field also provides "reverse converters"—#resource converters that convert from XRT/field-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 *XrtFldSetReverseConverters()*. This procedure registers the reverse converters. Every reverse converter allocates the space necessary to store the conversion result.

The following example shows how to retrieve a resource string once *XrtFldSetReverseConverters()* has been called:

```
Boolean
getStringValue(Widget w, String resource_name,
    String resource_type, String *string)
{
    XtArgVal value = 0;
    XrmValue from, to;

    XtVaGetValues(w, resource_name, &value, NULL);
    from.addr = (XtPointer) &value;
    from.size = sizeof(XtArgVal);
    to.addr = NULL;
    if (!XtConvertAndStore(w, resource_type, &from,
        XtRString, &to))
        return (False);
    *string = to.addr;
    return (True);
}
```

The returned string must be freed after use.

4.19 Internationalization

XRT/field conforms to the POSIX locale standard for internationalization. This ensures that users of XRT/field widgets can work in the language and symbol set with which they are familiar (provided, of course, that the appropriate operating system support is provided).

Specifying a Locale

To specify that your program is to use locale-specific information, and to set or retrieve the locale to use, call the *setlocale()* procedure. This procedure is defined as follows:

```
char *setlocale(int category, const char *locale);
```

The first argument, *category*, specifies the locale-specific information you wish to access.

The possible categories are:

LC_COLLATE	character sorting order for this locale
LC_CTYPE	character classification (ctype functions) for this locale
LC_MESSAGES	the values for yes and no answers for this locale, and the laguage in which messages are to be displayed
LC_MONETARY	the currency format for this locale
LC_NUMERIC	how numbers are displayed in this locale
LC_TIME	date and time information for this locale
LC_ALL	all of the above categories

The second argument, *locale*, specifies the locale to use. If *locale* is NULL, *setlocale()* uses the current locale. If *locale* is "C" or "POSIX", *setlocale()* sets the current locale to the C locale, which specifies an American-based, English-language environment that uses the ASCII code set. (This locale is also sometimes known as the POSIX locale, as this is the default locale specified by the IEEE POSIX committee.)

If locale is an empty string, *setlocale()* uses the locale specified by the LC_ALL environment variable, if it is defined. If LC_ALL is not defined, *setlocale()* uses the information specified by the *category* environment variable, where *category* is the category passed as an argument to *setlocale()*. If this environment variable is not defined, *setlocale()* uses the value of the LANG environment variable as the locale.

For example: when the call

```
setlocale(LC_COLLATE, "");
```

is executed, your program first checks whether LC_ALL is defined (contains the current locale). If it is not, the program checks whether the environment variable LC_COLLATE is defined. If LC_COLLATE does not exist, the program checks whether the locale is defined by the LANG environment variable. (If none of these environment variables are defined, the C locale is used.)

setlocale() returns the current locale (regardless of whether it has just been set or not). If an error occurs – for example, if the desired locale is not available – *setlocale()* returns NULL.

The *setlocale()* procedure must be called by your program before it manipulates any locale-specific data (or calls any library procedures that do so, such as *sort()*). Normally, *setlocale()* is called as the first statement in the *main()* procedure.

The most common use of *setlocale()* is as follows:

```
(void)setlocale(LC_ALL, "");
```

Specifying LC_ALL as the *category* argument ensures that all locale-specific information becomes available to your program.

Printing Locale Information

The XRT/field distribution includes a program, *printlocale*, located in *$XRTHOME/src/field/demos/locale*, which prints information about the current locale. If a particular piece of locale-specific information is not defined for the current locale, *printlocale* flags it as invalid.

printlocale allows you to ensure that your local environment contains the locale-specific information you are looking for. Locale-specific information is usually supplied by your vendor as a patch to your operating system. To manually set locale-specific data, use the *XrtFldSetDefaultLocaleStruct()* procedure, which is described in Appendix B on page 129.

Further Reading

A useful reference on locales and internationalization is *Programming for the World* by Sandra Martin O'Donnell, published by PTR Prentice Hall.

Part II

Reference Appendices

A

Resource Reference

Widget Synopses ■ *XrtField Resources*
XmText Resource Summary ■ *Inherited XrtField Resources*
XrtComboBox Resources ■ *XrtSpinBox Resources*
Inherited XrtComboBox and XrtSpinBox Resources ■ *XrtCalendar Resources*

This appendix lists all of the XRT/field resources in alphabetical order. Listed after the resource name is its data type, default value and a list of the procedures that 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 Synopses

The include file for each widget is *<Xm/XrtField.h>*.

XrtComboBox	Class Name:	XmXrtComboBox
	Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtFldManager → XmXrtComboBox
	Class Pointer:	xmXrtComboBoxWidgetClass
XrtCurrencyField	Class Name:	XmXrtCurrencyField
	Class Hierarchy:	Core → Primitive → XmText → XmXrtField → XmXrtFloatField → XmXrtcurrencyField
	Class Pointer:	xmXrtCurrencyFieldWidgetClass

XrtDateField	Class Name:	XmXrtDateField
	Class Hierarchy:	Core → Primitive → XmText → XmXrtField → XmXrtDateField
	Class Pointer:	xmXrtDateFieldWidgetClass
XrtFloatField	Class Name:	XmXrtFloatField
	Class Hierarchy:	Core → Primitive → XmText → XmXrtField → XmXrtFloatField
	Class Pointer:	xmXrtFloatFieldWidgetClass
XrtIntField	Class Name:	XmXrtIntField
	Class Hierarchy:	Core → Primitive → XmText → XmXrtField → XmXrtIntField
	Class Pointer:	xmXrtIntFieldWidgetClass
XrtSpinBox	Class Name:	XmXrtSpinBox
	Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtFldManager → XmXrtSpinBox
	Class Pointer:	xmXrtSpinBoxWidgetClass
XrtStringField	Class Name:	XmXrtStringField
	Class Hierarchy:	Core → Primitive → XmText → XmXrtField → XmXrtStringField
	Class Pointer:	xmXrtStringFieldWidgetClass
XrtCalendar	Class Name:	XmXrtCalendar
	Class Hierarchy:	Core → Composite → Constraint → XmManager → XmXrtFldManager → XmXrtCalendar
	Class Pointer:	xmXrtCalendarWidgetClass

A.2 XrtField Resources

This section lists the resources for the XrtField widget class and its subclasses. The following symbols precede resources defined for a particular field subclass:

- **C** currency fields (XrtCurrencyField)
- **D** date fields (XrtDateField)
- **F** float fields (XrtFloatField)
- **I** integer fields (XrtIntField)
- **S** string fields (XrtStringField)

XmNxrtFldAllowNull **Boolean** **True** **CSG**

Indicates whether a field is valid when empty. When set to `False`, an empty field is treated as invalid.

String fields are empty when they contain `NULL` or the empty string. Other types of fields are considered empty when they contain `NULL` or whitespace characters.

When `XmNxrtFldDefaultValue` is set, the field is not considered empty.

XmNxrtFldAutoComplete **Boolean** **False** **CSG**

Indicates whether to use auto-match completion or not when a picklist has been set.

Multi-character search is supported. When a sequence of keys are pressed, the current item in the picklist is set to the element whose initial characters match the key sequence. Each keypress refines the search.

D XmNxrtFldCalendar **Widget** **dynamic** **G**

If `XmNxrtFldPopupType` is of type `XRTFLD_POPUP_TYPE_CALENDAR`, this resource specifies the widget ID of the XrtCalendar widget. This resource is read-only.

D XmNxrtFldCalendarCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks called when the *Calendar()* action is invoked or an XrtComboBox's arrow button is pressed, assuming `XmNxrtComboType` is `XRTCOMBO_TYPE_CALENDAR`. Each callback is passed a pointer to the following structure:

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    struct tm       *date;
    Boolean         doit;       /* initially True */
} XrtFldCalendarCallbackStruct;
```

reason is either `XRTFLD_REASON_CALENDAR_BEGIN` (before the calendar popup has been displayed) or `XRTFLD_REASON_CALENDAR_END` (after the calendar popup has been dismissed). *date* is a pointer to a time structure which stores the date and time selected. If the popup does not include the time spinbox, then the time fields will be filled in with zeros. *date* can only be changed if *reason* is `XRTFLD_REASON_CALENDAR_END`. The behaviour of *doit* depends on the value of *reason*. If *reason* is `XRTFLD_REASON_CALENDAR_END` and *doit* is set to `False`, the selected date is disallowed.

XmNxrtFldCase **XrtFldCase** **XRTFLD_CASE_AS_IS** **CSG**

Specifies the case of alphabetic characters in a field. Valid values are:

`XRTFLD_CASE_AS_IS`	No conversion
`XRTFLD_CASE_UPPER`	Convert to upper case
`XRTFLD_CASE_LOWER`	Convert to lower case

XRT/field uses the `toupper` and `tolower` ANSI macros to convert characters. These macros may not work the same way on all locales.

XmNxrtFldCompleteCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that complete the value after the user exits the field. A pointer to an `XrtFldCompleteCallbackStruct` is passed to each callback.The form of the callback is:

```
void
CallbackProcedure(
    Widget      field;
    XtPointer   client_data,
    XtPointer   call_data /* points to an XrtFldCompleteCallbackStruct */
    )
```

where `XrtFldCompleteCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    String          value;
} XrtFldCompleteCallbackStruct;
```

reason is `XRTFLD_REASON_COMPLETE`. *value* is the future value of the field, in the widget's edit format (specified by `XmNxrtFldEditFormat` if it is defined for that subclass). Completion is performed by modifying this value. The value can also be changed by freeing the value and allocating and filling a new one.

To add callbacks to this list, use *XtAddCallback()*.

XmNxrtFldConvertCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks that convert the value stored in a field before validation takes place. A pointer to an `XrtFldConvertCallbackStruct` is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget      field;
    XtPointer   client_data,
    XtPointer   call_data   /* points to an XrtFldConvertCallbackStruct */
    )
```

where `XrtFldConvertCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason        reason;                         /* read-only */
    XEvent              *event;                         /* read-only */
    XrtFldAction        action;                         /* read-only */
    XmTextPosition      startPos, endPos;               /* read-only */
    XmTextPosition      currInsert;                     /* read-only */
    XmTextPosition      newInsert;
    XmTextBlock         text;
    String              value;
} XrtFldConvertCallbackStruct;
```

reason is always `XRTFLD_REASON_CONVERT`. *action* is the action that is currently being processed; it is one of `XRTFLD_ACTION_KEYSTROKE` (keystroke action), `XRTFLD_ACTION_PASTE` (paste action), `XRTFLD_ACTION_EXIT` (user exit action) or `XRTFLD_ACTION_SETVALUE` (programmatic change action).

text contains the text to be converted. Its contents depend on the value of *action*. If *action* is `XRTFLD_ACTION_KEYSTROKE`, *text* contains the last character typed by the user. If *action* is `XRTFLD_ACTION_PASTE`, *text* contains the text produced by the last paste or drag and drop operation. If *action* is `XRTFLD_ACTION_EXIT` or `XRTFLD_ACTION_SETVALUE`, *text* contains the entire contents of the field.

startPos contains the number of characters between the start of the field and the first character in *text*. *endPos* contains the number of characters between the start of the field and the last character in *text*. If *text* contains a single character or if *action* is `XRTFLD_ACTION_PASTE`, *startPos* and *endPos* are equivalent.

currInsert specifies the current position of the cursor. *newInsert* specifies the desired new position of the cursor. (By default, this is the same as *currInsert*.)

value contains the entire value. Its format depends on *action*. If a user exit action, a paste action or a keystroke action triggered the callback, *value* is provided in the field's edit format (specified by `XmNxrtFldEditFormat` if it is defined for that subclass). If a programmatic change action triggered the callback, *value* is provided in the field's display format (specified by `XmNxrtFldDisplayFormat` if it is defined for that subclass).

To add callbacks to this list, use *XtAddCallback()*.

G	**XmNxrtFldCounterIncrement**	**double**	**1**	**CSG**
D	**XmNxrtFldCounterIncrement**	**struct tm ***	**NULL**	**CSG**
F	**XmNxrtFldCounterIncrement**	**double**	**1**	**CSG**
I	**XmNxrtFldCounterIncrement**	**long**	**1**	**CSG**

Controls the amount to increase or decrease the arrow button value when the field is contained in an XrtSpinBox (whose `XmNxrtFldSpinType` resource is set to `XRTSPIN_TYPE_COUNTER`).

For date fields, `XmNxrtFldCounterIncrement` is set to `NULL`, and the increment is one unit of the smallest time unit currently displayed. (For example, if the date is displayed in the format `hh:mm`, the increment becomes one minute.) To set a counter increment for date fields, set the appropriate element(s) of the *struct tm* structure, and set all other elements to zero.

Remember to use *XrtFldDoubleToArgVal()* when setting resources of type *double*.

XmNxrtFldCreatePopups	**Boolean**	**False**	**C**

Indicates whether XRT/field should create popups immediately or on-demand. As would be expected, on-demand will conserve resources until they're needed.

C	**XmNxrtFldDecimalPlaces**	**int**	**XRTFLD_LOCALE**	**CSG**
F	**XmNxrtFldDecimalPlaces**	**int**	**2**	**CSG**

Specifies the number of decimal places to display. This resource is ignored when `XmNxrtFldDisplayFormat` is `XRTFLD_FORMAT_GENERAL`. If this resource is set to `XRTFLD_LOCALE`, the locale value is used.

The number of decimal places can be any value between 0 and 99. If a number larger than 99 is specified, 99 is assumed.

XmNxrtFldDefaultBackground	**Pixel**	**XRTFLD_UNCHANGED_COLOR**	**CSG**

Specifies the background color of a field displaying a valid default value (`XmNxrtFldIsDefault` is `True`). When set to `XRTFLD_UNCHANGED_COLOR`, the background color is the same as `XmNbackground`. On a monochrome server, this color is displayed as Black.

If the specified default value is invalid, the `XmNxrtFldErrorBackground` color is used for display.

XmNxrtFldDefaultForeground	**Pixel**	**XRTFLD_UNCHANGED_COLOR**	**CSG**

Specifies the foreground color of a field displaying a valid default value (`XmNxrtFldIsDefault` is `True`). When set to `XRTFLD_UNCHANGED_COLOR`, the default

foreground color is the same as XmNforeground. On a monochrome server, this color is displayed as White.

If the specified default value is invalid, the XmNxrtFldErrorForeground color is used for display.

C	**XmNxrtFldDefaultValue**	**double**	**0**	**CSG**
D	**XmNxrtFldDefaultValue**	**struct tm ***	**the current time**	**CSG**
F	**XmNxrtFldDefaultValue**	**double**	**0**	**CSG**
I	**XmNxrtFldDefaultValue**	**long**	**0**	**CSG**
S	**XmNxrtFldDefaultValue**	**String**	**NULL**	**CSG**

Specifies the value to be stored in the field when XmNvalue is set to be the empty field (either programmatically or by the user deleting all characters). This value is processed and validated as if it had been set by the program.

No default value is stored if the field:

- Is a date field with XmNxrtFldDefaultValue set to NULL.
- Is a float or currency field with XmNxrtFldDefaultValue set to XRTFLD_NOVALUE_FLOAT.
- Is an integer field with XmNxrtFldDefaultValue set to XRTFLD_NOVALUE.
- Is a string field with XmNxrtFldDefaultValue set to NULL.

In this case, the value of the field remains empty when the value is deleted. (The XmNxrtFldAllowNull resource allows you to flag empty fields as invalid.)

Remember to use *XrtFldDoubleToArgVal()* when setting resources of type *double*.

C	XmNxrtFldDisplayFormat	XrtFldFormat	XRTFLD_FORMAT_CURRENCY	CSG
F	XmNxrtFldDisplayFormat	XrtFldFormat	XRTFLD_FORMAT_GENERAL	CSG
I	XmNxrtFldDisplayFormat	XrtFldFormat	XRTFLD_FORMAT_GENERAL	CSG

Specifies a display format for an unmasked field, using locale values. If this resource is set to XRTFLD_FORMAT_DEFAULT, the edit format is used. If no display format or edit format is specified, the value is displayed as typed by the user.

For integer fields, the following formats are available:

XRTFLD_FORMAT_BINARY	Binary value (the 0b prefix is added to the display, as in 0b10110)
XRTFLD_FORMAT_GENERAL	Uses XmNxrtFldDisplayFormatString value
XRTFLD_FORMAT_HEX	Hex value (the 0x prefix is added, as in 0x52ab3)
XRTFLD_FORMAT_OCTAL	Octal value (the 0o prefix is added, as in 0o377)
XRTFLD_FORMAT_THOUSANDS	1,234

For float fields, the following formats are available:

XRTFLD_FORMAT_FIXED	Example: 12.34
XRTFLD_FORMAT_GENERAL	Uses XmNxrtFldDisplayFormatString value
XRTFLD_FORMAT_PERCENT	12.34%
XRTFLD_FORMAT_SCIENTIFIC	1.23e+02
XRTFLD_FORMAT_THOUSANDS	1,234.00

For currency fields, the float field formats can be used, plus the following:

XRTFLD_FORMAT_CURRENCY	$1,234.00
XRTFLD_FORMAT_CURRENCY_INTL	USD0 1,234.00

For both float and currency fields, the XmNxrtFldDecimalPlaces resource determines how many decimal places to display.

C	XmNxrtFldDisplayFormatString	String	NULL	CSG
D	XmNxrtFldDisplayFormatString	String	NULL	CSG
F	XmNxrtFldDisplayFormatString	String	NULL	CSG
I	XmNxrtFldDisplayFormatString	String	NULL	CSG

For date fields, this resource specifies the format in which to display the date and/or time. Date formats consist of any combination of the following:

D	numeric day of month (1-31)
DD	numeric day of month with leading zero
W	abbreviated weekday name
WW	weekday name
J	Julian date (day number of year) (1-366)
JJ	Julian date with leading zeroes
M	numeric month of year
MM	numeric month with leading zero
MMM	abbreviated month name
MMMM	month name
Y, YY	year within current century (00-99)
YYYY	year, including century
H	hour (0-12)
HH	0-12 hour with leading zero
h	hour (0-23)
hh	0-23 hour with leading zero
m	minute (0-59)
mm	minute with leading zero
s	second (0-59)
ss	second with leading zero
A or P	AM/PM representation
Z	time zone abbreviation
\\	next character is a literal

All other characters are treated as literals, and are displayed without processing.

If `XmNxrtFldDisplayFormatString` is set to `NULL`, the first string from the default `XmNxrtFldEnterFormatList` is used as the `XmNxrtFldDisplayFormatString`. See page 103 for the default `XmNxrtFldEnterFormatList`.

For float and integer fields, this resource specifies the format in which to display the value if `XmNxrtFldDisplayFormat` is `XRTFLD_FORMAT_GENERAL`. If non-`NULL`, this is used as the *sprintf* format string. Otherwise, "`%g`" is used for float fields, and "`%d`" is used for integer fields.

XmNxrtFldDisplayHorizScrollBar **XrtFldDispScrollBar** **XRTFLD_DISPSB_NEVER** **CSG**

Specifies when to display a horizontal scrollbar. Valid values are:

`XRTFLD_DISPSB_ALWAYS`	Scrollbar always displayed
`XRTFLD_DISPSB_NEVER`	Scrollbar never displayed
`XRTFLD_DISPSB_AS_NEEDED`	Scrollbar displayed when the value does not fit in the field

XmNxrtFldDisplayVertScrollBar **XrtFldDispScrollBar** **XRTFLD_DISPSB_AS_NEEDED** **CSG**

Specifies when to display a vertical scrollbar. Valid values are:

XRTFLD_DISPSB_ALWAYS	Scrollbar always displayed
XRTFLD_DISPSB_NEVER	Scrollbar never displayed
XRTFLD_DISPSB_AS_NEEDED	Scrollbar displayed when the value does not fit in the field

C **XmNxrtFldEditFormat** **XrtFldFormat** **XRTFLD_FORMAT_FIXED** **CSG**
F **XmNxrtFldEditFormat** **XrtFldFormat** **XRTFLD_FORMAT_GENERAL** **CSG**
I **XmNxrtFldEditFormat** **XrtFldFormat** **XRTFLD_FORMAT_GENERAL** **CSG**

Specifies the format in which to display the field value during editing. Setting this resource to XRTFLD_FORMAT_DEFAULT causes no change. This resource is ignored if a mask has been set.

For integer fields, valid values are:

XRTFLD_FORMAT_BINARY	Binary value (the 0b prefix is added to the display, as in 0b10110)
XRTFLD_FORMAT_GENERAL	Uses printf %d format (decimal)
XRTFLD_FORMAT_HEX	Hex value (the 0x prefix is added, as in 0x52ab3)
XRTFLD_FORMAT_OCTAL	Octal value (the 0o prefix is added, as in 0o377)
XRTFLD_FORMAT_THOUSANDS	1,234

For float fields, valid values are:

XRTFLD_FORMAT_FIXED	Example: 12.34
XRTFLD_FORMAT_GENERAL	Uses printf %g format (fixed or scientific, whichever is shorter)
XRTFLD_FORMAT_PERCENT	12.34%
XRTFLD_FORMAT_SCIENTIFIC	1.23e+02
XRTFLD_FORMAT_THOUSANDS	1,234.00

For currency fields, the float field formats can be used, plus the following:

XRTFLD_FORMAT_CURRENCY	$1,234.00
XRTFLD_FORMAT_CURRENCY_INTL USDO	1,234.00

When the user exits, the value will be displayed as specified by XmNxrtFldDisplayFormat.

XmNxrtFldEnterCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks invoked when the field gets input focus. A pointer to an `XrtFldEnterCallbackStruct` is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget          field;
    XtPointer       client_data,
    XtPointer       call_data /* points to an XrtFldEnterCallbackStruct */
    )
```

where `XrtFldEnterCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason    reason;         /* read-only */
    XEvent          *event;         /* read-only */
    String          value;          /* read-only */
    XmString        prompt_string;  /* read-only */
} XrtFldEnterCallbackStruct;
```

reason is always `XRTFLD_REASON_ENTER`. *prompt_string* is the current value of the `XmNxrtFldPromptString` resource. *value* is the value that will be displayed when the user edits the field, and is the `XmNxrtFldInternalValue` converted to edit format.

To add callbacks to this list, use *XtAddCallback()*.

D **XmNxrtFldEnterFormatList** **String *** **see below** **CSG**

Specifies a `NULL`-terminated list of valid date formats. The first item in the list is used as the `XmNxrtFldEditFormat`. To ensure that the displayed output can always be interpreted, the `XmNxrtFldDisplayFormatString` is added to the end of the list.

A date format consists of one or more of the following:

D	numeric day of month (1-31)
DD	numeric day of month with leading zero
W	abbreviated weekday name
WW	weekday name
M	numeric month of year (1-12)
MM	numeric month of year with leading zero
MMM	abbreviated month name
MMMM	month name
Y, YY	year within current century (00-99)
YYYY	year including century
H	hour (0-12)
HH	0-12 hour with leading zero
h	hour (0-23)
hh	hour with leading zero

m	minute (0-59)
mm	minute with leading zero
s	second (0-59)
ss	second with leading zero
A, P	AM/PM representation
Z	time zone abbreviation
\\	next character is a literal

All other characters are treated as literals, and are skipped when processing the input string.

The following is the default list of date formats:

YYYY-MM-DD hh:mm:ss

YYYY/MM/DD hh:mm:ss

DD/MM/YY hh:mm:ss

DD/MM/YYYY hh:mm:ss

DD-MM-YY hh:mm:ss

DD-MM-YYYY hh:mm:ss

MMM DD, YYYY hh:mm:ss

hh:mm:ss

hh:mm A

<locale's date/time format>

<locale's time format>

XmNxrtFldErrorBackground **Pixel** **XRTFLD_ERROR_COLOR** **CSG**

Specifies the background color of a field for which `XmNxrtFldIsValid` is `False`. If the color is `XRTFLD_ERROR_COLOR`, the background is red. On a monochrome terminal, the color is always displayed as black. If this resource is set to `XRTFLD_UNCHANGED_COLOR`, the widget background color is not changed when an error is detected.

XmNxrtFldErrorCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks invoked when an error is detected during validation or when an invalid field loses focus. A pointer to an `XrtFldErrorCallbackStruct` is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget          field;
    XtPointer       client_data,
    XtPointer       call_data /* points to an XrtFldErrorCallbackStruct */
    )
```

where `XrtFldErrorCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason        reason;                          /* read-only */
    XEvent              *event;                          /* read-only */
    int                 text_reason;                     /* read-only */
    XrtFldAction        action;                          /* read-only */
    XrtFldError         error;
    Boolean             is_modified;
    XmTextPosition      startPos, endPos;                /* read-only */
    XmTextPosition      currInsert;                      /* read-only */
    XmTextPosition      newInsert;
    XmTextBlock         text;
    XmString            error_string;                    /* read-only */
    Boolean             display;                  /* initially False */
    Boolean             traverse;                  /* initially True */
} XrtFldErrorCallbackStruct;
```

reason is XRTFLD_REASON_ERROR. *action* is the action that is currently being processed; it is one of XRTFLD_ACTION_KEYSTROKE (keystroke action), XRTFLD_ACTION_EXIT (user exit action) or XRTFLD_ACTION_SETVALUE (programmatic change action).

text_reason indicates which XmText callback triggered this callback, and is one of:

XmCR_ACTIVATE	triggered by XmNactivateCallback
XmCR_LOSING_FOCUS	triggered by XmNlosingFocusCallback
XmCR_MODIFYING_TEXT_VALUE	triggered by XmNmodifyVerifyCallback
XmCR_VALUE_CHANGED	triggered by XmNvalueChangedCallback

error indicates which validation operation detected the invalid value. Valid values are:

XRTFLD_ERROR_EXCEED_LENGTH	The length of a line of the value is greater than the value stored in XmNxrtFldMaxLineLength.
XRTFLD_ERROR_EXCEED_LINES	The number of lines in the value is greater than the value stored in XmNxrtFldMaxLines.
XRTFLD_ERROR_IS_NULL	The field is empty when XmNxrtFldAllowNull is set to False.
XRTFLD_ERROR_INVALID_CHAR	A character in the value is either in the XmNxrtFldInvalidChars list, or is not in the XmNxrtFldValidChars list.
XRTFLD_ERROR_INVALID_DATATYPE	A character in the value is not appropriate for the data type of the field (for example, an alphabetic character in an integer field).
XRTFLD_ERROR_NOMATCH_MASK	The value does not match the mask specified by XmNxrtFldMask.
XRTFLD_ERROR_NOMATCH_PICKLIST	The value is not included in the pick list specified by XmNxrtFldPickList.
XRTFLD_ERROR_OUT_OF_RANGE	The value is either larger than the upper limit specified by XmNxrtFldMax, or is smaller than the lower limit specified by XmNxrtFldMin.

XRTFLD_ERROR_VALIDATECB	The value has been flagged as invalid by a validation callback. (The validation callback does this by setting its *doit* structure member to False.)

To override any error except XRTFLD_ERROR_INVALID_DATATYPE, set *error* to XRTFLD_ERROR_NONE.

is_modified contains the current value of XmNxrtFldIsModified. Changing *is_modified* changes the value of this resource.

The *currInsert*, *newInsert*, *startPos*, *endPos* and *text* members are identical to their corresponding members in the XrtFldConvertCallbackStruct structure.

error_string is the message to be displayed by the application when an error occurs, and is the value of the XmNxrtFldErrorString resource. This message is of type XmString (compound string).

The *display* member is only used if *action* is XRTFLD_ACTION_KEYSTROKE. It indicates whether the erroneous value should be displayed by the field. Setting *display* to True tells XRT/field to display the value. By default, *display* is False, and the value is not displayed (and the terminal beeps if XmNverifyBell is True).

The *traverse* member is defined if *action* is XRTFLD_ACTION_EXIT. It allows you to control whether the user is allowed to use Return or Tab to traverse out of a field when it contains an error. By default, *traverse* is True and traversal takes place. Setting *traverse* to False means that the user remains in the field even when he or she presses Return or Tab.

To add callbacks to this list, use *XtAddCallback()*.

XmNxrtFldErrorForeground **Pixel** **XRTFLD_ERROR_COLOR** **CSG**

Specifies the foreground color of a field for which XmNxrtFldIsValid is False. If the color is XRTFLD_ERROR_COLOR, the foreground is white. On a monochrome terminal, the color is always displayed as white. If this resource is set to XRTFLD_UNCHANGED_COLOR, the widget foreground color is not changed when an error is detected.

XmNxrtFldErrorString **XmString** **NULL** **CSG**

Specifies the message displayed to the user on an error. This resource is for use by the application in an error callback, and is not used by XRT/field.

XmNxrtFldExitCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks to be called when the user exits a valid field. A pointer to an XrtFldExitCallbackStruct is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget          field;
    XtPointer       client_data,
    XtPointer       call_data   /* points to an XrtFldExitCallbackStruct */
    )
```

where XrtFldExitCallbackStruct is defined as:

```
typedef struct {
    XrtFldReason        reason;       /* read-only */
    XEvent              *event;       /* read-only */
    int                 text_reason;  /* read-only */
    Boolean             is_modified;
    String              value;        /* read-only */
    Boolean             traverse;     /* initially True */
} XrtFldExitCallbackStruct;
```

Here, *reason* is XRTFLD_REASON_EXIT. *event* is the X event that caused the callback. *value* is the future value of the field (the value to be displayed). This value is displayed in the field's display format (specified by XmNxrtFldDisplayFormat).

text_reason indicates which XmText callback triggered this exit callback. Valid values are:

XmCR_LOSING_FOCUS	triggered by XmNlosingFocusCallback
XmCR_ACTIVATE	triggered by XmNactivateCallback

text_reason is only set to XmCR_ACTIVATE when the user attempts to exit the field by pressing the Return key. All other exit methods set *text_reason* to XmCR_LOSING_FOCUS.

is_modified contains the current value of XmNxrtFldIsModified. Changing *is_modified* changes the value of this resource.

traverse indicates whether the user should be allowed to traverse out of the field using the Return or Tab key. By default, *traverse* is True, and traversal takes place.

To add callbacks to this list, use *XtAddCallback()*.

C **XmNxrtFldInternalValue** **double** **XRTFLD_MAXDOUBLE** **CSG**
D **XmNxrtFldInternalValue** **struct tm *** **NULL** **CSG**
F **XmNxrtFldInternalValue** **double** **XRTFLD_MAXDOUBLE** **CSG**
I **XmNxrtFldInternalValue** **long** **XRTFLD_MAXLONG** **CSG**
S **XmNxrtFldInternalValue** **String** **NULL** **CSG**

Specifies the current value of the field (the last complete value entered). The default values shown above are used if XmNvalue cannot be converted to the appropriate data type. If the field was empty, the value of XmNxrtFldDefaultValue is used.

Setting this resource triggers a programmatic change action: the value specified is validated, and conversions are performed if provided.

Remember to use *XrtFldDoubleToArgVal()* when setting resources of type *double*.

D **XmNxrtFldInternalValueLong** **long** **XRTFLD_MAXLONG** **CSG**

Specifies the current value of a date field as a *long*. If the current value cannot be represented as a *long* (i.e., it is not in a range roughly between 1902 and 2033), this resource is set to `XRTFLD_MAXLONG`.

XmNxrtFldInvalidChars **String** **NULL** **CSG**

Specifies a string of characters that may not be part of the value. If `XmNxrtFldValidChars` has also been set, the invalid string is searched first.

XmNxrtFldIsDefault **Boolean** **dynamic** **G**

Determines whether the field is currently empty (and therefore contains the default value).

XmNxrtFldIsModified **Boolean** **dynamic** **CSG**

Indicates how the value was last modified. If `True`, the user last changed the value stored in the field. If `False`, the program last changed the value stored in the field, or it has not been changed at all.

This resource can be set by changing the *is_modified* field in an error callback or exit callback structure.

XmNxrtFldIsValid **Boolean** **dynamic** **G**

Indicates the result of the last validation process. If `XmNxrtFldIsValid` is `True`, no error was detected when the last action was processed (keystroke action, user exit action, or programmatic change action).

XmNxrtFldLastError **XrtFldError** **dynamic** **CSG**

Specifies the last error that occurred. If the field is currently valid, this resource is set to `XRTFLD_ERROR_NONE`.

C **XmNxrtFldLeadingZero** **Boolean** **True** **CSG**
F **XmNxrtFldLeadingZero** **Boolean** **True** **CSG**

Specifies whether numbers in the range (-1,1) should be displayed with a `0` preceding the decimal point.

XmNxrtFldMask **String** **NULL** **CSG**

For string and date fields, this resource specifies a mask to be used to restrict input to a position-dependent set of characters, and to denote editable versus non-editable character positions. (This resource cannot be used with other subclasses.) The data being entered or displayed is formatted using the mask to limit the characters accepted, and to visually cue the user to the required data type. Each character in the mask acts as a placeholder.

The following mask characters are defined:

#	0-9, minus, comma, plus, decimal point
@	0-9
H	hex (0-9, A-F)
U	alpha, converted to uppercase
L	alpha, converted to lowercase
A	alpha, not converted
*	any character
?	any character in `XmNxrtFldValidChars`
\\	next character is a literal

Any other character is treated as a literal and is displayed as a cue, even when the field's value is empty. If a proportional font is used, the literals will shift position as the user types, unless the empty positions are initially filled using the `XmNvalue` resource.

Overstrike mode is set by default in a masked field (the `XmNxrtFldOverstrike` value is ignored). Deleting a character will not shift characters to the left, but will replace the previous character with a space.

If `XmNvalue` is set in a masked field, characters will be read from left to right, and placed even if the data does not exactly match the mask. Literals in the mask will be skipped. For example, the value "`(416)555-1212`" could be set in a field whose mask is "*(###)###-####*". This allows flexibility in setting values from a database, and allows the user to perform X copy/paste operations.

If `XmNxrtFldMask` is specified, datatype validation is not performed.

XmNxrtFldMaskChars **String** **"#@HULA*?\\"** **CSG**

Specifies the characters that are to replace the predefined `XmNxrtFldMask` characters. The characters must appear in the following order:

#	The number (digit, minus sign, comma or decimal point) special character
@	The digit special character
H	The hexadecimal digit special character
U	The upper case alphabetic special character
L	The lowercase alphabetic special character
A	The alphabetic special character
*	The generic special character (which permits any character to appear)
?	The special character indicating a character in `XmNxrtFldValidChars`
\\	The escape character

If fewer characters are provided, the remainder will be set to their default values.

XmNxrtFldMatchMask **XrtFldMatchMask** **XRTFLD_MATCH_ALL** **CSG**

Specifies whether the value must match the entire mask. Valid values are:

XRTFLD_MATCH_ALL	must match entire mask
XRTFLD_MATCH_PARTIAL_L_TO_R	allow left-to-right partial matching
XRTFLD_MATCH_PARTIAL_R_TO_L	allow right-to-left partial matching

A mask is treated as partially matched if the unmatched portion of the value consists entirely of spaces.

XmNxrtFldMatchPickList **Boolean** **True** **CSG**

Specifies whether a value must match one of the items in XmNxrtFldPickList.

C XmNxrtFldMax **double** **dynamic** **CSG**
D XmNxrtFldMax **struct tm *** **NULL** **CSG**
F XmNxrtFldMax **double** **dynamic** **CSG**
I XmNxrtFldMax **long** **dynamic** **CSG**
S XmNxrtFldMax **String** **NULL** **CSG**

Specifies the largest valid value for a field.

For float and integer fields, the default value for XmNxrtFldMax is the upper limit of the range specified by the XmNxrtFldRange resource.

In string fields, the *strcoll()* procedure is used to compare the value in the field with the largest valid value specified by XmNxrtFldMax.

Remember to use *XrtFldDoubleToArgVal()* when setting resources of type *double*.

XmNxrtFldMaxLineLength **int** **XRTFLD_MAXINT** **CSG**

Specifies the maximum number of characters per line. If XmNwordWrap is False, entering a character beyond this limit triggers the error callbacks.

XmNxrtFldMaxLineLength overrides the XmText resource XmNmaxLength.

XmNxrtFldMaxLines **int** **XRTFLD_MAXINT** **CSG**

Specifies the maximum number of lines, meaningful only if XmNeditMode is XmMULTI_LINE_EDIT. Entering a character beyond this limit triggers the error callbacks.

XmNxrtFldMenuAlignment **XrtFldMenuAlignment** **XRTFLD_MENU_ALIGN_END** **CSG**

Specifies the menu alignment relative to the ComboBox text field. Valid values are:

XRTFLD_MENU_ALIGN_BEGINNING	Align left side of menu with left side of text field
XRTFLD_MENU_ALIGN_END	Align right side of menu with right side of text field
XRTFLD_MENU_ALIGN_BOTH	Both sides of menu aligned with text field; menu and text field identical in size

XmNxrtFldMenuCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks called when the user invokes the *Menu()* action or presses an XrtComboBox's arrow button. A pointer to an `XrtFldMenuCallbackStruct` is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget          field;
    XtPointer       client_data,
    XtPointer       call_data   /* points to an XrtFldMenuCallbackStruct */
    )
```

where `XrtFldMenuCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason        reason;     /* read-only */
    XEvent              *event;     /* read-only */
    int                 position;   /* read-only */
    String              value;
    Boolean             doit;       /* initially True */
} XrtFldMenuCallbackStruct;
```

reason is either `XRTFLD_REASON_MENU_BEGIN` (before the menu has been displayed) or `XRTFLD_REASON_MENU_END` (after the menu has been displayed). *position* is the index within the pick list (starting from 1); if the user has not selected an item, *position* is set to `XRTFLD_NOVALUE`. *value* is the value that will be displayed in the field; it may be changed if *reason* is `XRTFLD_REASON_MENU_END`.

The behavior of *doit* depends on the value of *reason*. If *reason* is `XRTFLD_REASON_MENU_BEGIN` and *doit* is set to `False`, the menu is not displayed (in this case, the application may perform a custom operation). If reason is `XRTFLD_REASON_MENU_END` and *doit* is set to `False`, the picked value is disallowed.

To add callbacks to this list, use *XtAddCallback()*.

XmNxrtFldMenuList **Widget** ***dynamic*** **G**

Specifies the widget ID of the pick list (contained in the popup menu). This resource is read-only.

The pick list is of class XmList.

D **XmNxrtFldMilleniumThreshold** **int** **0** **CSG**

Specifies how to interpret 2-digit year values with respect to the year 2000. Valid values are between 0 and 100. When 0, 2-digit years are prefixed by 1900. Otherwise, 2-digit years less than the value of this resource are prefixed by 2000 and 1900 prefixes 2-digit years greater than or equal to this resource. Thus, if the value is 70, 2-digit years less than 70 are prefixed by 2000, and 2-digit years greater than or equal to 70 are prefixed by 1900.

G	XmNxrtFldMin	double	dynamic	CSG
D	XmNxrtFldMin	struct tm *	NULL	CSG
F	XmNxrtFldMin	double	dynamic	CSG
I	XmNxrtFldMin	long	dynamic	CSG
S	XmNxrtFldMin	String	NULL	CSG

Specifies the smallest valid value for a field.

For float and integer fields, the default value for `XmNxrtFldMin` is the lower limit of the range specified by the `XmNxrtFldRange` resource.

In string fields, the *strcoll()* procedure is used to compare the value in the field with the smallest valid value specified by `XmNxrtFldMin`.

Remember to use *XrtFldDoubleToArgVal()* when setting resources of type *double*.

C	XmNxrtFldNegativeFormat	XrtFldNegFormat	XRTFLD_NEG_LOCALE	CSG
F	XmNxrtFldNegativeFormat	XrtFldNegFormat	XRTFLD_NEG_AS_IS	CSG
I	XmNxrtFldNegativeFormat	XrtFldNegFormat	XRTFLD_NEG_AS_IS	CSG

Specifies how negative values are to be displayed. The following negative formats are available:

`XRTFLD_NEG_AS_IS`	display with leading dash
`XRTFLD_NEG_PARENTHESES`	remove dash and enclose in parentheses

Currency fields can also use the `XRTFLD_NEG_LOCALE` format, which specifies that the locale setting should be used.

If the value stored in a float field or currency field is displayed as a percentage (using the `XRTFLD_FORMAT_PERCENT` format) and `XRTFLD_NEG_PARENTHESES` is specified as the negative display format, the parentheses, when they appear, enclose the percent sign.

XmNxrtFldOverstrike	Boolean	dynamic	CSG

Specifies whether the field is in overstrike mode. This value is ignored in masked fields (which always operate in overstrike mode).

C	XmNxrtFldPickList	double *	NULL	CSG
D	XmNxrtFldPickList	struct tm **	NULL	CSG
F	XmNxrtFldPickList	double *	NULL	CSG
I	XmNxrtFldPickList	int *	NULL	CSG
S	XmNxrtFldPickList	String *	NULL	CSG

Specifies a list of suggested values. If the widget is a string field or date field, the list of values must be terminated by `NULL`. If the widget is an integer field or float field, the value of the first element must be the remaining number of elements in the list.

If `XmNvalue` must match one of the items, set `XmNxrtFldMatchPickList` to `True`.

XmNxrtFldPickListIndex	int	dynamic	CSG

Specifies the position of the current item within the pick list specified by `XmNxrtFldPickList`, if applicable. The first item in the list is at position 0. Setting the value to greater than the number of items in the list displays the last item. If no item from the list is displayed, the value is set to `XRTFLD_NOVALUE`.

XmNxrtFldPopupType **XrtFldPopupType** **dynamic** **CG**

Specifies which type of popup to use when the user presses CTRL Down-Arrow from an XRT/field widget or presses the arrow button in a combo box. For XrtDateField, the default is `XRTFLD_POPUP_CALENDAR`. For all other field types, the default is a `XRTFLD_POPUP_PICKLIST`.

Note that this resource is inherited by all XRT/field subclasses (available to all), but should only be used for the XrtDateField, as no other supported implementations currently exist.

XmNxrtFldPromptString **XmString** **NULL** **CSG**

Specifies the message displayed for the user when the field gets focus. This resource is for use by the application by procedures in the `XmNxrtFldEnterCallback` list, and is not used by the widget.

C XmNxrtFldRange **XrtFldRange** **XRTFLD_RANGE_DOUBLE** **CSG**
F XmNxrtFldRange **XrtFldRange** **XRTFLD_RANGE_DOUBLE** **CSG**
I XmNxrtFldRange **XrtFldRange** **XRTFLD_RANGE_INT** **CSG**

Specifies the default range of values that can be stored in the widget (unless `XmNxrtFldMax` or `XmNxrtFldMin` is set).

For float fields, the following ranges can be specified:

`XRTFLD_RANGE_FLOAT`	single-precision floating-point
`XRTFLD_RANGE_DOUBLE`	double-precision floating-point (default)

For currency fields, the range of values that can be stored is always assumed to be double-precision floating-point, regardless of the value of `XmNxrtFldRange`.

For integer fields, the following ranges can be specified. The numeric ranges listed here assume a 32-bit machine. Check the file */usr/include/limits.h* for the exact ranges for your machine.

`XRTFLD_RANGE_CHAR`	signed character (range -128 to 127)
`XRTFLD_RANGE_SHORT`	short signed integer (range -32768 to 32767)
`XRTFLD_RANGE_INT`	standard-sized signed integer (default; range -2147483648 to 2147483647)
`XRTFLD_RANGE_LONG`	long signed integer (same as `XRTFLD_RANGE_INT`)
`XRTFLD_RANGE_UNSIGNED_CHAR`	unsigned character (range 0 to 255)
`XRTFLD_RANGE_UNSIGNED_SHORT`	short unsigned integer (range 0 to 65535)
`XRTFLD_RANGE_UNSIGNED_INT`	standard-sized unsigned integer (range 0 to 4294967295)
`XRTFLD_RANGE_UNSIGNED_LONG`	long unsigned integer (same as `XRTFLD_RANGE_UNSIGNED_INT`)

XmNxrtFldSearchPolicy **XrtFldSearchPolicy** **XRTFLD_SEARCH_SINGLE** **CSG**

Specifies the popup menu search behaviour when the end-user presses one or more keys. Valid values are:

XRTFLD_SEARCH_SINGLE	Only single-character searching is supported. Each time the end-user presses a key, the current item is set to the first (or next) element whose initial character matches the key. Each keypress restarts the search.
XRTFLD_SEARCH_MULTICHAR	Multi-character searching is supported. When the end-user presses a sequence of keys, the current item is set to the element whose initial characters match the key sequence. Each keypress refines the search.

XmNxrtFldSpinCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks to be called when the user invokes a *Spin()* action or presses an XrtSpinBox arrow button. A pointer to an XrtFldSpinCallbackStruct is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget          field;
    XtPointer       client_data,
    XtPointer       call_data   /* points to an XrtFldSpinCallbackStruct */
    )
```

where XrtFldSpinCallbackStruct is defined as:

```
typedef struct {
    XrtFldReason        reason;                         /* read-only */
    XEvent              *event;                         /* read-only */
    XrtFldDirection     direction;                      /* read-only */
    Boolean             at_boundary;                    /* read-only */
    Boolean             doit;                       /* initially True */
} XrtFldSpinCallbackStruct;
```

If no spin callback is specified, the next item in the XmNxrtFldPickList is displayed.

reason is XRTFLD_REASON_SPIN_BEGIN or XRTFLD_REASON_SPIN_END. *direction* is XRTFLD_DIRECTION_PREVIOUS or XRTFLD_DIRECTION_NEXT. *at_boundary* is True if the parent's XmNxrtFldSpinType resource is set to XRTSPIN_TYPE_PICKLIST and the current value is at the beginning or the end of the pick list. *doit* should be set to False if *reason* is XRTFLD_REASON_SPIN_BEGIN and incrementing is not to occur.

To add callbacks to this list, use *XtAddCallback()*.

D XmNxrtFldSpinPolicy **XrtFldSpinPolicy** **XRTFLD_SPIN_FIELD** **CSG**

Specifies the spin policy of a field in a spin box when the spin type is XRTSPIN_TYPE_COUNTER. Valid values are:

XRTFLD_SPIN_FIELD	Field behaves normally (for a spinbox)
XRTFLD_SPIN_SUBFIELD	Date subfield where the cursor currently is spins (i.e. the hours of a time field).
XRTFLD_SPIN_WRAP	Same as XRTFLD_SPIN_FIELD, except that it allows a wrap around up from max to min, or down from min to max.

XmNxrtFldValidateCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks invoked as the last step of the validation process. A pointer to an `XrtFldValidateCallbackStruct` is passed to each callback. The form of the callback is:

```
void
CallbackProcedure(
    Widget      field;
    XtPointer   client_data,
    XtPointer   call_data /* points to an XrtFldValidateCallbackStruct */
    )
```

where `XrtFldValidateCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason        reason;                          /* read-only */
    XEvent              *event;                          /* read-only */
    XrtFldAction        action;                          /* read-only */
    int                 num_calls;                       /* read-only */
    XmTextPosition      startPos, endPos;                /* read-only */
    XmTextPosition      currInsert;                      /* read-only */
    XmTextPosition      newInsert;
    XmTextBlock         text;
    Boolean             doit;                       /* initially True */
    String              value;
} XrtFldValidateCallbackStruct;
```

reason is `XRTFLD_REASON_VALIDATE`. *event* is the X event that caused the callback. *action* is the action that is currently being processed; it is one of `XRTFLD_ACTION_KEYSTROKE` (keystroke action), `XRTFLD_ACTION_PASTE` (paste action), `XRTFLD_ACTION_EXIT` (user exit action) or `XRTFLD_ACTION_SETVALUE` (programmatic change action).

The *currInsert*, *newInsert*, *startPos*, *endPos* and *text* members are identical to their corresponding members in the `XrtFldConvertCallbackStruct` structure.

value contains the entire value stored in the field. This value includes any conversions or completions previously specified.

The format of *value* depends on *action*. If the validation callback is triggered by a user exit action, a paste action or a keystroke action, *value* is provided in the field's edit format (specified by `XmNxrtFldEditFormat` if it is defined for that subclass). If the validation callback is triggered by a programmatic change action, *value* is provided in the field's display format (specified by `XmNxrtFldDisplayFormat` if it is defined for that subclass).

doit indicates whether the value stored in *text* should be treated as valid. If the callback routine determines that the value is invalid, the routine must set *doit* to `False`.

A callback routine can modify the value stored in *text* or *value* if desired. To do this, the routine must free the storage pointed to by *text* or *value*, allocate storage for the new value, fill the storage, and point *text* or *value* to the new storage.

If *text* or *value* is modified, the entire editing process (conversion, completion when appropriate, and validation) is performed again, including this validation callback. *num_calls* is incremented each time the validation callback is called.

If both *text* and *value* are modified, *value* takes precedence.

To add callbacks to this list, use *XtAddCallback()*.

XmNxrtFldValidChars **String** **NULL** **CSG**

Specifies a string of acceptable characters that may be entered. If `XmNxrtFldInvalidChars` has also been set, the invalid string is searched first for a given character.

XmNxrtFldVisibleItemCount **int** **10** **CSG**

Specifies the maximum visible number of items in the drop-down menu displayed by the *Menu()* action. If the number of items is greater than this value, a scrollbar is displayed.

A.3 XmText Resource Summary

Fields inherit all XmText resources. The following summarizes these resources for your convenience. Consult your Motif documentation for more complete information.

XmNactivateCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked when the Return key is pressed in a widget (and `XmNeditMode` is `XmSINGLE_LINE_EDIT`).

XmNautoShowCursorPosition **Boolean** **True** **CSG**

Specifies whether text will scroll if necessary to display the cursor.

XmNblinkRate **int** **500** **CSG**

The time (in milliseconds) the cursor spends visible or invisible before blinking.

XmNcolumns **short** **dynamic** **CSG**

The number of text characters displayed in the text window.

XmNcursorPosition **XmTextPosition** **0** **CSG**

The location of the cursor in the XmText widget.

XmNcursorPositionVisible **Boolean** **True** **CSG**

Specifies whether the cursor is visible.

XmNeditable **Boolean** **True** **CSG**

Specifies whether the user can edit the text string.

XmNeditMode **int** **XmSINGLE_LINE_EDIT** **CSG**

Specifies whether the user can enter multiple lines of text.

XmNfocusCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked when the XmText widget gets input focus.

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

The font list used for the widget text.

XmNgainPrimaryCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked when the XmText widget obtains ownership of the primary selection.

XmNlosePrimaryCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked when the XmText widget loses ownership of the primary selection.

XmNlosingFocusCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked when the XmText widget loses input focus.

XmNmarginHeight **Dimension** **5** **CSG**

The space between the top and bottom edges of the widget and its text.

XmNmarginWidth **Dimension** **5** **CSG**

The space between the left and right edges of the widget and its text.

XmNmaxLength **int** **largest integer** **CSG**

The maximum length of the text string.

XmNmodifyVerifyCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked before the text string is changed.

XmNmotionVerifyCallback **XtCallbackList** **NULL** **CSG**

Callbacks invoked before the cursor is moved.

XmNpendingDelete **Boolean** **True** **CSG**

Specifies whether selected text is to be deleted as soon as the next text insertion occurs.

XmNresizeHeight **Boolean** **False** **CSG**

Specifies whether the text widget will expand vertically to display multi-line text strings.

XmNresizeWidth **Boolean** **False** **CSG**

Specifies whether the text widget will expand horizontally to display long text lines.

XmNrows **short** **dynamic** **CSG**

The number of lines of text currently displayed.

XmNselectionArray **XtPointer** **default array** **CSG**

An array representing the actions resulting from multiple mouse clicks.

XmNselectionArrayCount **int** **4** **CSG**

The number of items in the array specified by `XmNselectionArray`.

XmNselectThreshold **int** **5** **CSG**

The number of pixels to drag the cursor to select the next character of the text string.

XmNsource **XmTextSource** **Default source** **CSG**

The source the XmText widget uses for displaying text.

XmNtopCharacter **XmTextPosition** **0** **CSG**

The position of the text currently displayed at the top of the window.

XmNvalue **String** **" "** **CSG**

The text string to display in the widget.

XmNvalueWcs	wchar_t *	(wchar_t *)""	CSG

The text string to display in the widget, defined as a pointer to a *wchar_t*. (This resource is not supported by XRT/field subclasses.)

XmNvalueChangedCallback	XtCallbackList	NULL	CSG

Callbacks invoked after the text string is changed.

XmNverifyBell	Boolean	True	CSG

Specifies whether a bell is to sound when a string verification produces no action.

XmNwordWrap	Boolean	False	CSG

Specifies whether lines are to be broken between words.

A.4 Inherited XrtField Resources

The following additional resources are inherited by XrtField widgets from their parent classes. The following table alphabetically lists the resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

Resource	Inherited From
XmNaccelerators	Core
XmNancestorSensitive	Core
XmNbackground	Core
XmNborderColor	Core
XmNborderPixmap	Core
XmNborderWidth	Core
XmNbottomShadowColor	Primitive
XmNbottomShadowPixmap	Primitive
XmNcolormap	Core
XmNdepth	Core
XmNdestroyCallback	Core
XmNforeground	Primitive
XmNheight	Core
XmNhighlightColor	Primitive
XmNhighlightOnEnter	Primitive
XmNhighlightPixmap	Primitive

Resource	Inherited From
XmNhighlightThickness	Primitive
XmNinitialResourcesPersistent	Core
XmNmappedWhenManaged	Core
XmNnavigationType	Primitive
XmNscreen	Core
XmNsensitive	Core
XmNshadowThickness	Primitive
XmNtopShadowColor	Primitive
XmNtopShadowPixmap	Primitive
XmNtranslations	Core
XmNtraversalOn	Primitive
XmNunitType	Primitive
XmNuserData	Primitive
XmNwidth	Core
XmNx	Core
XmNy	Core

A.5 XrtComboBox Resources

This section lists the XrtComboBox resources in alphabetical order.

XmNxrtComboArrow **Widget** **dynamic** **G**

The widget ID of the child arrow button.

XmNxrtComboArrowClass **WidgetClass** **xmArrowButtonWidgetClass** **C**

The widget class of the child arrow button. This resource can only be set at creation time.

XmNxrtComboArrowType **XrtFldArrowType** **XRTFLD_ARROW_WEDGE** **C**

Specifies the appearance of the child arrow button. This resource is only used when `XmNxrtComboArrowClass` is set to `xmDrawnButtonWidgetClass`. Valid values are:

`XRTFLD_ARROW_NONE`	Arrow button displayed with no pixmap
`XRTFLD_ARROW_WEDGE`	Arrow contains pixmap similar to those found in Windows 95
`XRTFLD_ARROW_WITH_UNDERBAR`	Arrow contains a pixmap similar to those found in Windows 3.1
`XRTFLD_ARROW_CUSTOM`	Arrow shape is application-specific

XmNxrtComboField **Widget** **dynamic** **G**

The widget ID of the child field.

XmNxrtComboFieldClass **WidgetClass** **NULL** **C**

The widget class of the child field. This resource can only be set when the XrtComboBox is created. If this resource is `NULL`, the `XmNxrtComboFieldType` value is used.

XmNxrtComboFieldType **XrtFldFieldType** **XRTFLD_FIELDTYPE_STRING** **C**

Specifies the class of the child field. Valid values are:

`XRTFLD_FIELDTYPE_CURRENCY`

`XRTFLD_FIELDTYPE_DATE`

`XRTFLD_FIELDTYPE_FLOAT`

`XRTFLD_FIELDTYPE_INT`

`XRTFLD_FIELDTYPE_STRING`

This resource can only be set when the XrtComboBox is created, and is ignored if `XmNxrtComboFieldClass` has been set.

XmNxrtComboSpacing **Dimension** **0** **CSG**

Specifies the space in pixels between the child field and the child arrow button.

A.6 XrtSpinBox Resources

This section lists the XrtSpinBox resources in alphabetical order.

XmNxrtSpinArrowLayout **XrtSpinArrowLayout** **XRTSPIN_ARROWS_SPLIT** **CSG**

Specifies the position of the arrows. Valid values are:

XRTSPIN_ARROWS_BEGINNING	up and down arrows at box left
XRTSPIN_ARROWS_END	up and down arrows at box right
XRTSPIN_ARROWS_SPLIT	left and right arrows at either end of box

XmNxrtSpinArrowOrientation **unsigned char** **XmHORIZONTAL** **CSG**

Specifies the orientation of the arrows. Valid values are:

XmHORIZONTAL	left and right arrows
XmVERTICAL	up and down arrows

If XmNxrtSpinArrowLayout is set to XRTSPIN_ARROWS_BEGINNING or XRTSPIN_ARROWS_END and XmNxrtSpinArrowOrientation is set to XmVERTICAL, the vertical arrows appear over each other.

XmNxrtSpinArrowType **XrtFldArrowType** **XRTFLD_ARROW_WEDGE** **C**

Specifies the appearance of the arrows. This resource is only used when XmNxrtSpinDecrArrowClass and XmNxrtSpinIncrArrowClass is is set to xmDrawnButtonWidgetClass. Valid values are:

XRTFLD_ARROW_NONE	Arrow button displayed with no pixmap
XRTFLD_ARROW_WEDGE	Arrow contains pixmap similar to those found in Windows 95
XRTFLD_ARROW_WITH_UNDERBAR	Arrow contains a pixmap similar to those found in Windows 3.1
XRTFLD_ARROW_CUSTOM	Arrow shape is application-specific

XmNxrtSpinDecrArrow **Widget** **dynamic** **G**

The widget ID of the child decrement arrow button (direction left or down).

XmNxrtSpinDecrArrowClass **WidgetClass** **xmArrowButtonWidgetClass** **C**

The widget class of the child decrement arrow button (direction left or down). This resource can only be set when the XrtSpinBox is created.

XmNxrtSpinField **Widget** **dynamic** **G**

The widget ID of the child field.

XmNxrtSpinFieldClass **WidgetClass** **NULL** **C**

The widget class of the child field. This resource can only be set when the ComboBox is created. If NULL, the XmNxrtComboFieldType value is used.

XmNxrtSpinFieldName **String** **"field"** **C**

The name of the child field. This resource can only be set when the XrtSpinBox is created.

XmNxrtSpinFieldType **XrtFldFieldType** **XRTFLD_FIELDTYPE_STRING** **C**

Specifies the class of the child field. Valid values are:

```
XRTFLD_FIELDTYPE_CURRENCY
XRTFLD_FIELDTYPE_DATE
XRTFLD_FIELDTYPE_FLOAT
XRTFLD_FIELDTYPE_INT
XRTFLD_FIELDTYPE_STRING
```

This resource can only be set when the XrtSpinBox widget is created, and is ignored if `XmNxrtSpinFieldClass` has been set.

XmNxrtSpinIncrArrow **Widget** **dynamic** **G**

The widget ID of the child increment arrow button (direction right or up).

XmNxrtSpinIncrArrowClass **WidgetClass** **xmArrowButtonWidgetClass** **C**

The widget class of the child increment arrow button (direction right or up). This resource can only be set when the XrtSpinBoxwidget is created.

XmNxrtSpinSpacing **Dimension** **0** **CSG**

Specifies the horizontal space in pixels between the field and arrow button child widgets.

XmNxrtSpinType **XrtSpinType** **XRTSPIN_TYPE_PICKLIST** **CSG**

The type of data in child fields. Valid values are:

`XRTSPIN_TYPE_COUNTER`	spin range defined by `XmNxrtFldMin` and `XmNxrtFldMax`; increment specified by `XmNxrtFldCounterIncrement`
`XRTSPIN_TYPE_PICKLIST`	spin items in `XmNxrtFldPickList`

A.7 Inherited XrtComboBox and XrtSpinBox Resources

The following resources are inherited by XrtComboBox and XrtSpinBox from their parent classes. The following table alphabetically lists the resources along with the parent class that defines them. Consult your Motif and Xt documentation for more information.

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

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

A.8 XrtCalendar Resources

This section lists the XrtCalendar resources in alphabetical order.

XmNxrtCalendarCellMargin **Dimension** **0** **CSG**

Specifies the margin space around each calendar cell.

XmNxrtCalendarCellHighlightThickness **Dimension** **1** **CSG**

Specifies the cell highlight thickness. The highlight is around the selected calendar cell.

XmNxrtCalendarCreateMonth **Boolean** **True** **C**

If this resource is set to `True`, the XrtCalendar widget will create a month spin box above the calendar portion of the XrtCalendar widget at the top left.

XmNxrtCalendarCreateTime **Boolean** **False** **C**

If this resource is set to `True`, the XrtCalendar widget will create a time spin box below the calendar portion of the XrtCalendar widget.

XmNxrtCalendarCreateYear **Boolean** **True** **C**

If this resource is set to `True`, the XrtCalendar widget will create a year spin box above the calendar portion of the XrtCalendar widget at the top right.

XmNxrtCalendarEnabled **Boolean** **True** **CSG**

Specifies whether or not user-interaction is allowed on the widget.

XmNxrtCalendarFirstDayofWeek **int** **0** **CSG**

Specifies what day of the week should be considered the first day of the week. The first day of the week is normally Sunday (day 0), but this can be changed.

XmNxrtCalendarFullDayNames **Boolean** **True** **CSG**

Specifies whether to use full or abbreviated day names. If `True`, the locale's full day name is used for the header.

XmNxrtCalendarHeaderNumChars **int** **1** **CSG**

Specifies the number of header characters displayed for each month.

XmNxrtCalendarHeaderBackground **Pixel** **dynamic** **CSG**

Specifies the color used for the header background. Unless changed, this resource will be identical to `XmNbackground`.

XmNxrtCalendarHeaderFontList **XmFontList** **dynamic** **CSG**

Specifies the font list used for the header. Unless changed, this resource will be identical to `XmNfontList`.

XmNxrtCalendarHeaderForeground **Pixel** **dynamic** **CSG**

Specifies the color used for the header foreground. Unless changed, this resource will be identical to `XmNbackground`.

XmNxrtCalendarMarginBottom **Dimension** **0** **CSG**

Specifies the margin at bottom of the XrtCalendar widget.

XmNxrtCalendarMarginLeft **Dimension** **0** **CSG**

Specifies the XrtCalendar widget's left margin.

XmNxrtCalendarMarginRight **Dimension** **0** **CSG**

Specifies the XrtCalendar widget's right margin.

XmNxrtCalendarMarginTop **Dimension** **0** **CSG**

Specifies the XrtCalendar widget's top margin.

XmNxrtCalendarMaxRollDate **struct tm *** **NULL** **CSG**

Specifies the maximum date that the user can select. The calendar widget does not permit a date that is past this date to be displayed.

XmNxrtCalendarMinRollDate **struct tm *** **NULL** **CSG**

Specifies the minimum date that the user can select. The calendar widget does not permit a date that is previous to this date to be displayed.

XmNxrtCalendarModifyCallback **XtCallbackList** **NULL** **CSG**

Specifies a list of callbacks called when the user modifies the date. A pointer to an `XrtCalendarModifyCallbackStruct` is passed to each callback with the appropriate reason. The form of the callback is:

```
void
CallbackProcedure(
   Widget      field;
   XtPointer   client_data,
   XtPointer   call_data /* points to XrtCalendarModifyCallbackStruct */
   )
```

where `XrtCalendarModifyCallbackStruct` is defined as:

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    struct tm       *date;
    Boolean         doit;       /* initially true */
} XrtCalendarModifyCallbackStruct;
```

The value of *reason* depends on how the calendar widget was changed. Depending on whether the date was changed with the spin month widget, the spin day widget, the spin time widget, the day grid or through set values, the *reason* can be `XRTFLD_REASON_CALENDAR_MONTH`, `XRTFLD_REASON_CALENDAR_YEAR`, `XRTFLD_REASON_CALENDAR_TIME`, `XRTFLD_REASON_CALENDAR_DAYGRID` or `XRTFLD_REASON_CALENDAR_SETVALUES`. *date* is a pointer to a time structure which stores the user-selected date and time. If *doit* is set to `False`, the selected date is disallowed.

XmNxrtCalendarSpecialDayBackground **Pixel** **dynamic** **CSG**

Specifies the color used for the special day background. Unless changed, this resource will be identical to `XmNbackground`.

XmNxrtCalendarSpecialDayForeground **Pixel** **dynamic** **CSG**

Specifies the color used for the special day foreground. Unless changed, this resource will be identical to `XmNforeground`.

XmNxrtCalendarSpecialDays **XrtCalendarSpecialDay ** NULL** **CSG**

This resource points to a list of special days defined for the widget. Each entry in this list sets a special day or a list of special days on the widget. These special days are highighted using the special day foreground and background color. This list must be `NULL` terminated.

XmNxrtCalendarSelectedBackground **Pixel** **dynamic** **CSG**

Specifies the color used for the selection background. Unless changed, this resource will be identical to `XmNforeground` (reverse video is the default for selection).

XmNxrtCalendarSelectedForeground **Pixel** **dynamic** **CSG**

Specifies the color used for the selection foreground. Unless changed, this resource will be identical to `XmNbackground` (reverse video is the default for selection).

XmNxrtCalendarValue **struct tm *** **the current time** **CSG**

Specifies the current value of the date on the calendar display. This is updated when the user selects a new date. Changing this resource has the side effect of changing `XmNxrtCalendarValueLong`.

XmNxrtCalendarValueLong **long** **the current time** **CSG**

Specifies the current value of the date on the calendar display as a *long*. If the current value cannot be represented as a *long* (i.e. it is not in a range roughly between 1902 and 2033, this resource is set to `XRTFLD_MAXLONG`. Changing this resource has the side effect of changing `XmNxrtCalendarValue`.

B

Procedures and Methods Reference

This appendix lists the XRT/field procedures and methods in alphabetical order.

XmCreateXrtCalendar()

Creates an XrtCalendar widget.

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

parent is the parent. *name* is the created widget's name. *arglist* is a list of *argcount* arguments to be applied to the widget. All children are managed.

XmCreateXrtComboBox()

Creates an XrtComboBox widget with a field and an arrow button as children. The name of the arrow button is `arrow`, and the name of the field is `field`. The widget ID of the combo box is returned.

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

parent is the parent. *name* is the created field's name. *arglist* is a list of *argcount* arguments to be applied to the field. All children are managed.

XmCreateXrtCurrencyField()

Creates a currency field.

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

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

XmCreateXrtDateField()

Creates a date field.

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

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

XmCreateXrtFloatField()

Creates a float field.

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

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

XmCreateXrtIntField()

Creates an integer field.

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

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

XmCreateXrtSpinBox()

Creates an XrtSpinBox widget with a field and two arrow buttons as children. The arrow names are `incr_arrow` and `decr_arrow`. If not specified, the name of the field is `field`. Additional fields (or other widgets) can be added to the XrtSpinBox by creating them as children. The widget ID of the spin box is returned.

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

parent is the parent. *name* is the created field's name. *arglist* is a list of *argcount* arguments to be applied to the field. All children are managed.

XmCreateXrtStringField()

Creates a string field.

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

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

XmIsXrtCalendar()

Returns `True` if a widget is an XrtCalendar widget.

```
Widget
XmIsXrtCalendar(
    Widget  widget
    )
```

XmIsXrtComboBox()

Returns `True` if a widget is an XrtComboBox widget.

```
int
XmIsXrtComboBox(
    Widget  widget
    )
```

XmIsXrtCurrencyField()

Returns `True` if a widget is a currency field or a subclass.

```
int
XmIsXrtCurrencyField(
    Widget  widget
    )
```

XmIsXrtDateField()

Returns `True` if a widget is a date field or a subclass.

```
int
XmIsXrtDateField(
    Widget  widget
    )
```

XmIsXrtFloatField()

Returns `True` if a widget is a float field or a subclass. `True` is also returned if the widget is a currency field.

```
int
XmIsXrtFloatField(
    Widget  widget
    )
```

XmIsXrtField()

Returns `True` if a widget is a subclass of an XrtField widget or a subclass.

```
int
XmIsXrtField(
    Widget  widget
    )
```

XmIsXrtIntField()

Returns `True` if a widget is an integer field or a subclass.

```
int
XmIsXrtIntField(
    Widget  widget
    )
```

XmIsXrtSpinBox()

Returns `True` if a widget is an XrtSpinBox widget.

```
int
XmIsXrtSpinBox(
    Widget  widget
    )
```

XmIsXrtStringField()

Returns `True` if a widget is a string field.

```
int
XmIsXrtStringField(
    Widget  widget
    )
```

XrtCalendarAddSpecialDay()

Adds a special day specification to the end of the special day list. It returns `True` if successful.

```
Boolean
XrtCalendarAddSpecialDay(
    Widget                  widget,
    XrtCalendarSpecialDay   *special_day
    )
```

XrtCalendarDeleteSpecialDay()

Deletes a special day specification from the special day list. The special day must be in the special day list and it must match the parameters used in the structure. It returns `True` if successful.

```
Boolean
XrtCalendarDeleteSpecialDay(
    Widget                  widget,
    XrtCalendarSpecialDay   *special_day
    )
```

XrtCalendarDupSpecialDays()

Duplicates a list of special days. It returns `NULL` if it fails.

```
XrtSpecialDay **
XrtCalendarDupSpecialDays(
XrtCalendarSpecialDay       **special_day_list
    )
```

XrtCalendarFreeSpecialDays()

Frees a list of special days. It returns `True` if successful.

```
Boolean
XrtCalendarFreeSpecialDays(
    XrtCalendarSpecialDay **special_day_list
    )
```

XrtFldCheckFields()

Traverses the child hierarchy of a manager widget, returning a structure containing three lists: a list of all fields, a list of invalid fields, and a list of defaulted fields. It returns `False` if no fields were found.

```
Boolean
XrtFldCheckFields(
    Widget          parent,
    XrtFldFieldData *data   /* returned */
    )
```

The list members of *data* should be freed after use by calling *XrtFldFieldDataFree()*.

The list of invalid fields includes fields whose invalid status has been overridden by a callback in the `XmNxrtFldValidateCallback` list.

XrtFldCvtLongToTime()

Converts a long integer to a *struct tm* structure. The long integer, *value*, is equivalent to a *time_t* structure, and contains the number of seconds relative to January 1, 1970. (Negative values for *value* are assumed to represent dates earlier than 1970.)

```
void
XrtFldCvtLongToTime(
    long        value,
    struct tm   *date   /* returned */
    )
```

XrtFldCvtStringToDate()

Converts a string to a *struct tm* structure. Returns `False` if the conversion failed.

```
Boolean
XrtFldCvtStringToDate(
    String      input,
    String      mask,
    int         millenium_threshold,
    struct tm   *result     /* returned */
    )
```

input is the string to be converted. *mask* is the date format to use when reading the string; the components of this format are the same as those used by the `XmNxrtFldEnterFormatList` resource. Two digit year dates less than *millenium_threshold* are interpreted as dates in the 21st century. For example, if *millenium_threshold* is 70 and *input* is 69, *XrtFldCvtStringToDate()* interprets 69 as 2069.

input can be "`TODAY`", which specifies today's date as of midnight, or "`NOW`", which specifies the current time.

XrtFldCvtStringToTime()

Converts a string to a *struct tm* structure. Returns `False` if the conversion failed.

```
Boolean
XrtFldCvtStringToTime(
    String      value,
    String      format,
    struct tm   *date       /* returned */
    )
```

value is the string to be converted. *format* is the date format to use when reading the string; the components of this format are the same as those used by the `XmNxrtFldEnterFormatList` resource.

value can be "`TODAY`", which specifies today's date as of midnight, or "`NOW`", which specifies the current time.

XrtFldCvtTimeToLong()

Converts a *struct tm* structure to a long integer equivalent to a *time_t* value (containing the number of seconds relative to January 1, 1970). Unlike values of type *time_t*, the value returned by this procedure can represent dates earlier than 1970 as negative numbers. The *tm_gmtoff* and *tm_zone* structure members are ignored.

```
Boolean
XrtFldCvtTimeToLong(
    struct tm   *date,
    long        *value      /* returned */
    )
```

This procedure returns `False` if any structure member is invalid, or if the date cannot fit into the long integer. (For a 32-bit machine, years between approximately 1902 to 2033 can be returned by this procedure.)

XrtFldDoubleToArgVal()

Converts a *double* to an XtArgVal, for use when setting certain float or currency field resource values.

```
XtArgVal
XrtFldDoubleToArgVal(
    double  value
    )
```

XrtFldFieldDataFree()

Frees the data allocated by *XrtFldCheckFields()*.

```
Boolean
XrtFldFieldDataFree(
    XrtFldFieldData *data
    )
```

XrtFldGetCurrencyLocale()

Gets the locale currency values currently in use (i.e. set by the LC_NUMERIC or LC_CURRENCY category value call to *setlocale()*, or by *XrtFldSetDefaultLocaleStruct()*).

```
void
XrtFldGetCurrencyLocale(
    struct lconv    *data
    )
```

data is the returned data; it is read-only, and must not be freed.

XrtFldGetDateLocale()

Gets the date/time locale values currently in use.

```
void
XrtFldGetDateLocale(
    XrtFldDTConv    *data
    )
```

data is the returned data; it is read-only, and must not be freed.

XrtFldGetNumericLocale()

Gets the locale numeric values currently in use (i.e. set by the LC_NUMERIC category value call to *setlocale()*, or by *XrtFldSetDefaultLocaleStruct()*).

```
void
XrtFldGetNumericLocale(
    struct lconv    *data
    )
```

data is the returned data; it is read-only, and must not be freed.

XrtFldGetDisplayValue()

Gets the string that will be displayed when the field is exited. The returned string should be freed after use.

```
String
XrtFldGetDisplayValue(
    Widget          field
    )
```

XrtFldGetField()

Gets a field of a specified class from the list of cached fields, or creates one if necessary. This procedure is designed for use with XRT/table, which displays only one text widget at a time. Using a per-table cache ensures that a minimal number of fields are created.

```
Widget
XrtFldGetField(
    Widget          parent,
    String          name,
    WidgetClass     field_class,
    Arg             *arglist,
    int             argcount
    )
```

parent is the parent table widget; *name* is the created field's name; *arglist* is a list of *argcount* arguments to be applied to the field.

XrtFldGetPassword()

Sets the characters within an `XmNxrtFldConvertCallback` structure to a specified *echo* character. Characters typed by the end user are swapped into the *string* parameter. *XrtFldGetPassword()* returns `False` if the widget is not an XrtField subclass. *XrtFldGetPassword()* is normally used inside an application's Convert callback for password entry fields.

```
Boolean
XrtFldGetPassword(
    Widget  field,
    String  *string,    /* returned */
    char    echo,
    XrtFldConvertCallbackStruct *cbs
    )
```

string is the current password, set as the user types. *string* must be initialized to `NULL`, and freed after use. *echo* is the character to be displayed in place of the keystroke; if *echo* is zero, the keystroke is echoed as typed. *cbs* is the Convert callback structure passed to you.

XrtFldMrmInitialize()

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

```
void
XrtFldMrmInitialize(void)
```

XrtFldSetDefaultLocaleStruct()

Sets the global locale data to be used if *setlocale()* has not been called or a given locale member is invalid. If locale information is not supported by the operating system, *dt_conv* allows you to supply it yourself.

This procedure returns `False` if any data is invalid.

```
Boolean
XrtFldSetDefaultLocaleStruct(
    struct lconv *locale,
    XrtFldDTconv dt_conv
    )
```

If *locale* and *dt_conv* are `NULL`, the US locale data is used.

XrtFldSetReverseConverters()

Registers XRT/field's reverse resource converters for use with *XtConvertAndStore()*. Reverse converters allow applications to get resource values in a human-readable form.

```
void
XrtFldSetReverseConverters(void)
```

Note that converters are only supplied for XRT/field datatypes.

XrtFldValidateValue()

Validates a field's current `XmNvalue` string, returning `False` if the value is invalid. If valid, the field's internal value (`XmNxrtFldInternalValue`) is set. This procedure would normally only be called after a keystroke action, since validation is automatically performed when the field is exited.

```
Boolean
XrtFldValidateValue(
    Widget  field
    )
```

C

Macros

The following macros are provided for convenience when converting time values. They are defined in *$XRTHOME/include/Xm/XrtField.h*.

Macro Name	Definition
`MIN_TO_SEC(m)`	`(60L * (m))`
Converts a minute value *m* to the equivalent number of seconds.	
`HOUR_TO_SEC(h)`	`(3600L * (h))`
Converts an hour value *h* to the equivalent number of seconds.	
`DAY_TO_SEC(d)`	`(86400L * (d))`
Converts a day *d* (a value from 1 to 366) to the equivalent number of seconds.	
`MONTH_TO_SEC(m)`	`(2592000L * (m))`
Converts a month *m* to the equivalent number of seconds.	
`YEAR_TO_SEC(y)`	`(31536000L * (y))`
Converts the year *y* to the equivalent number of seconds.	

D

Translations and Actions Reference

Default Translations
Action Routines

D.1 Default Translations

XrtField widgets inherit all XmText translations, and define the following translations of their own:

	Event	Action
All Fields	<Key>osfCancel	Cancel()
	Ctrl <Key>s	Scrollbars()
	Ctrl <Key>S	Scrollbars()
	Ctrl <Key>osfDown	Menu(Display)
	Ctrl <Key>osfUp	Menu(Hide)
	Ctrl <Key>osfDown	Calendar(Display)
	Ctrl <Key>osfUp	Calendar(Hide)
Fields in an XrtSpinBox	<Key>osfRight	Spin(Next)
	<Key>osfUp	Spin(Next)
	<Key>osfLeft	Spin(Previous)
	<Key>osfDown	Spin(Previous)

Consult your Motif documentation for more details on XmText translations.

D.2 Action Routines

XRT/field defines the following action routines:

Calendar()
Controls the display of a popup XrtCalendar widget for an XrtDateField widget.

This action takes one parameter:

`Display`	Display date popup.
`Hide`	Hide date popup.

Holding down the CTRL key and pressing the down-arrow toggles the calendar popup display for the XrtDateField.

Cancel()
Replaces the value displayed in the field (the value currently being edited by the user) with the value of `XmNxrtFldInternalValue`. This value normally appears in the field's edit format (specified by `XmNxrtEditFormat` if it is defined for that subclass).

Complete()
Completes the value from the contents of the `XmNxrtFldPickList` (if a non-ambiguous item can be found). If no list has been specified for a numeric field, zeroes are used. If no list has been specified for a date field, the current date is used. If the completion cannot be performed, the terminal beeps.

Menu()
Displays the `XmNxrtFldPickList` items in a popup menu. Each item is displayed in the field's edit format.

This action takes one parameter:

`Display`	Display popup menu
`Hide`	Hide popup menu

The user can change the value stored in a field by selecting from this popup menu. Changing the value in this way triggers a programmatic change action.

NoAction()
Does nothing. Setting an event to call this routine effectively disables the translation.

Overstrike()
Sets the insert/overstrike behavior (specified by the `XmNxrtFldOverstrike` resource).

This action takes one parameter:

`False`	Turn off overstriking
`True`	Turn on overstriking
`Toggle`	Switch overstrike mode to opposite of current mode

Scrollbars()

Toggles the scrollbar display. The first call to *Scrollbars()* turns scrollbar display off by setting both `XmNxrtFldDisplayHorizScrollBar` and `XmNxrtFldDisplayVertScrollBar` to `XRTFLD_DISPSB_NEVER`.

The second call to *Scrollbars()* turns scrollbar display on by setting `XmNxrtFldDisplayHorizScrollBar` to `XRTFLD_DISPSB_ALWAYS`, and setting `XmNxrtFldDisplayVertScrollBar` to `XRTFLD_DISPSB_AS_NEEDED`.

Subsequent calls to *Scrollbars()* toggle scrollbar display off and on, as described above.

Spin()

Displays the next item in the `XmNxrtFldPickList` or increments the current value, depending on the value of the XrtSpinBox's `XmNxrtSpinType` resource.

This action takes one parameter:

`Next`	Display next item or increment value
`Prev`	Display previous item or decrement value

E

Data Types

This appendix lists the XRT/field data types in alphabetical order.

XrtCalendarModifyCallbackStruct

Structure passed to callbacks on the `XmNxrtCalendarModifyCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    struct tm       *date;
    Boolean         doit;
} XrtCalendarModifyCallbackStruct;
```

XrtCalendarReason

Enumeration used in callbacks to specify the callback reason:

```
XRTFLD_REASON_CALENDAR_MONTH
XRTFLD_REASON_CALENDAR_YEAR
XRTFLD_REASON_CALENDAR_TIME
XRTFLD_REASON_CALENDAR_DAYGRID
XRTFLD_REASON_CALENDAR_SETVALUES
```

XrtCalendarSpecialDay

Structure defining special days.

```
typedef struct {
    XrtCalendarSpecialDayType   type;

/* the rest of the structure is interpreted based on the type */

    int     day;
    int     day_of_week;    /* 0 == Sun && 6 == Sat */
    int     week;           /* 1 based */
    int     month;          /* 0 == Jan and 11 == Dec */
    int     year;           /* Full year. 1998 not 98 */
} XrtCalendarSpecialDay;
```

XrtCalendarSpecialDayType

Enumeration used to specify the type of special day to be added or removed.

```
XRTCALENDAR_SPECIAL_DAY_BY_DATE
XRTCALENDAR_SPECIAL_DAY_BY_DAY_OF_WEEK
```

XrtFldAction

Enumeration used in a callback to specify the action (event) that triggered it:

```
XRTFLD_ACTION_EXIT
XRTFLD_ACTION_KEYSTROKE
XRTFLD_ACTION_SETVALUE
XRTFLD_ACTION_PASTE
```

XrtFldArrowType

Enumeration used by XrtComboBox widgets with the `XmNxrtComboArrowType` resource to specify the type of arrow displayed:

```
XRTFLD_ARROW_CUSTOM
XRTFLD_ARROW_NONE
XRTFLD_ARROW_WEDGE
XRTFLD_ARROW_WITH_UNDERBAR
```

XrtFldCalendarCallbackStruct

Structure passed to a Calendar callback on the field widget.

```
typedef struct {
    XrtFldReason    reason;     /* read-only */
    XEvent          *event;     /* read-only */
    struct tm       *date;
    Boolean         doit;       /* initially true */
} XrtFldCalendarCallbackStruct;
```

XrtFldCase

Enumeration used to specify case conversion with `XmNxrtFldCase`:

```
XRTFLD_CASE_AS_IS
XRTFLD_CASE_LOWER
XRTFLD_CASE_UPPER
```

XrtFldCompleteCallbackStruct

Structure passed to callbacks on the `XmNxrtFldCompleteCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    String          value;
} XrtFldCompleteCallbackStruct;
```

XrtFldConvertCallbackStruct

Structure passed to callbacks on the `XmNxrtFldConvertCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    XrtFldAction    action;
    XmTextPosition  startPos, endPos;
    XmTextPosition  currInsert;
    XmTextPosition  newInsert;
    XmTextBlock     text;
    String          value;
} XrtFldConvertCallbackStruct;
```

XrtFldDirection

Enumeration passed to `XmNxrtFldSpinCallback` callbacks to specify spin or counter direction:

```
XRTFLD_DIRECTION_NEXT
XRTFLD_DIRECTION_PREVIOUS
```

XrtFldDispScrollBar

Enumeration used by `XmNxrtFldDisplayHorizScrollBar` and `XmNxrtFldDisplayVertScrollBar` to specify scrollbar display:

```
XRTFLD_DISPSB_ALWAYS
XRTFLD_DISPSB_AS_NEEDED
XRTFLD_DISPSB_NEVER
```

XrtFldDTconv

Structure defining date/time information, passed to *XrtFldSetDefaultLocaleStruct()* in environments that do not support locales:

```
typedef struct {
    String abbrev_month_names[12];
    String month_names[12];
    String abbrev_weekday_names[7];
    String weekday_names[7];
    String time_format;
    String sdate_format;
    String dtime_format;
    String am_string;
    String pm_string;
    String ldate_format;
    String abbrev_tz;
} XrtFldDTconv;
```

XrtFldEnterCallbackStruct

Structure passed to callbacks on the `XmNxrtFldEnterCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    String          value;
    XmString        prompt_string;
} XrtFldEnterCallbackStruct;
```

XrtFldError

Enumeration used (by callbacks on the `XmNxrtFldErrorCallback` list) to specify errors:

```
XRTFLD_ERROR_EXCEED_LENGTH
XRTFLD_ERROR_EXCEED_LINES
XRTFLD_ERROR_INVALID_CHAR
XRTFLD_ERROR_INVALID_DATATYPE
XRTFLD_ERROR_IS_NULL
XRTFLD_ERROR_NOMATCH_MASK
XRTFLD_ERROR_NOMATCH_PICKLIST
XRTFLD_ERROR_OUT_OF_RANGE
XRTFLD_ERROR_VALIDATECB
XRTFLD_ERROR_NONE
```

XrtFldErrorCallbackStruct

Structure passed to callbacks on the `XmNxrtFldErrorCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    int             text_reason;
    XrtFldAction    action;
    XrtFldError     error;
    Boolean         is_modified;
    XmTextPosition  startPos, endPos;
    XmTextPosition  currInsert;
    XmTextPosition  newInsert;
    XmTextBlock     text;
    XmString        error_string;
    Boolean         display;
    Boolean         traverse;
} XrtFldErrorCallbackStruct;
```

XrtFldExitCallbackStruct

Structure passed to callbacks on the `XmNxrtFldExitCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    int             text_reason;
    Boolean         is_modified;
    String          value;
    Boolean         traverse;
} XrtFldExitCallbackStruct;
```

XrtFldFieldData

Structure used by *XrtFldCheckFields()* to return lists of descendant field widgets:

```
typedef struct {
    Widget  *field_list;
    int     num_field_list;
    Widget  *invalid_list;
    int     num_invalid_list;
    Widget  *default_list;
    int     num_default_list;
} XrtFldFieldData;
```

XrtFldFieldType

Enumeration used by `XmNxrtComboFieldType` and `XmNxrtSpinFieldType` to specify the subclass of the child field:

```
XRTFLD_FIELDTYPE_CURRENCY
XRTFLD_FIELDTYPE_DATE
XRTFLD_FIELDTYPE_FLOAT
XRTFLD_FIELDTYPE_INT
XRTFLD_FIELDTYPE_NONE
XRTFLD_FIELDTYPE_STRING
```

XrtFldFormat

Enumeration used by `XmNxrtFldEditFormat` and `XmNxrtFldDisplayFormat` to specify edit and display formats:

```
XRTFLD_FORMAT_BINARY
XRTFLD_FORMAT_CURRENCY
XRTFLD_FORMAT_CURRENCY_INTL
XRTFLD_FORMAT_DEFAULT
XRTFLD_FORMAT_FIXED
XRTFLD_FORMAT_GENERAL
XRTFLD_FORMAT_HEX
XRTFLD_FORMAT_OCTAL
XRTFLD_FORMAT_PERCENT
XRTFLD_FORMAT_SCIENTIFIC
XRTFLD_FORMAT_THOUSANDS
```

XrtFldMatchMask

Enumeration used by `XmNxrtFldMatchMask` to specify whether partial mask matching is allowed:

```
XRTFLD_MATCH_ALL
XRTFLD_MATCH_PARTIAL_L_TO_R
XRTFLD_MATCH_PARTIAL_R_TO_L
```

XrtFldMenuAlignment

Enumeration used by `XmNxrtFldMenuAlignment`:

```
XRTFLD_MENU_ALIGN_BEGINNING
XRTFLD_MENU_ALIGN_END
XRTFLD_MENU_ALIGN_BOTH
```

XrtFldMenuCallbackStruct

Structure passed to callbacks on the `XmNxrtFldMenuCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    int             position;
    String          value;
    Boolean         doit;
} XrtFldMenuCallbackStruct;
```

XrtFldNegFormat

Enumeration used by `XmNxrtFldNegativeFormat` to specify negative number display format:

```
XRTFLD_NEG_AS_IS
XRTFLD_NEG_LOCALE
XRTFLD_NEG_PARENTHESES
```

XrtFldPopupType

Enumeration used to control the type of popup for a field or combo box.

```
XRTFLD_POPUP_PICKLIST
XRTFLD_POPUP_CALENDAR
```

XrtFldRange

Enumeration used by `XmNxrtFldRange` to specify field ranges:

```
XRTFLD_RANGE_CHAR
XRTFLD_RANGE_DEFAULT
XRTFLD_RANGE_DOUBLE
XRTFLD_RANGE_FLOAT
XRTFLD_RANGE_INT
XRTFLD_RANGE_LONG
XRTFLD_RANGE_SHORT
XRTFLD_RANGE_UNSIGNED_CHAR
XRTFLD_RANGE_UNSIGNED_INT
XRTFLD_RANGE_UNSIGNED_LONG
XRTFLD_RANGE_UNSIGNED_SHORT
```

XrtFldReason

Enumeration used in callbacks to specify the callback reason:

```
XRTFLD_REASON_COMPLETE
XRTFLD_REASON_CONVERT
XRTFLD_REASON_ENTER
XRTFLD_REASON_ERROR
XRTFLD_REASON_EXIT
XRTFLD_REASON_MENU_BEGIN
XRTFLD_REASON_MENU_END
XRTFLD_REASON_SPIN_BEGIN
XRTFLD_REASON_SPIN_END
XRTFLD_REASON_VALIDATE
XRTFLD_REASON_CALENDAR_BEGIN
XRTFLD_REASON_CALENDAR_END
```

XrtFldSearchPolicy

Enumeration used by the `XmNxrtFldSearchPolicy` resource to specify search behavior in pick lists:

```
XRTFLD_SEARCH_SINGLE
XRTFLD_SEARCH_MULTICHAR
```

XrtFldSpinCallbackStruct

Structure passed to callbacks on the `XmNxrtFldSpinCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    XrtFldDirection direction;
    Boolean         at_boundary;
    Boolean         doit;
} XrtFldSpinCallbackStruct;
```

XrtFldSpinPolicy

Enumeration used to control the spin type for spin boxes whose type is `XRTSPIN_TYPE_COUNTER`.

```
XRTFLD_SPIN_FIELD
XRTFLD_SPIN_SUBFIELD
XRTFLD_SPIN_WRAP
```

XrtFldValidateCallbackStruct

Structure passed to callbacks on the `XmNxrtFldValidateCallback` list:

```
typedef struct {
    XrtFldReason    reason;
    XEvent          *event;
    XrtFldAction    action;
    int             num_calls;
    XmTextPosition  startPos, endPos;
    XmTextPosition  currInsert;
    XmTextPosition  newInsert;
    XmTextBlock     text;
    Boolean         doit;
    String          value;
} XrtFldValidateCallbackStruct;
```

XrtSpinArrowLayout

Enumeration used by XrtSpinBox widgets to specify arrow button positions:

```
XRTSPIN_ARROWS_BEGINNING
XRTSPIN_ARROWS_END
XRTSPIN_ARROWS_SPLIT
```

XrtSpinType

Enumeration used by XrtSpinBox widgets to specify type:

```
XRTSPIN_TYPE_COUNTER
XRTSPIN_TYPE_PICKLIST
```

F

Resource Types and Classes

Common XrtField Resources ■ *XrtStringField Resources*
XrtIntField Resources ■ *XrtFloatField Resources*
XrtDateField Resources ■ *XrtComboBox Resources*
XrtSpinBox Resources ■ *XrtCalendar Resources*

This appendix lists the resource type and resource class for each XRT/field resource. The resources common to all XrtField subclasses are listed first, followed by the resources defined for each subclass and the resources defined for XrtComboBox, XrtSpinBox and XrtCalendar widgets.

F.1 Common XrtField Resources

Name	Class	Type
XmNxrtFldAllowNull	XmCBoolean	XmRBoolean
XmNxrtFldCase	XmCXrtFldCase	XmRxrtFldCase
XmNxrtFldCompleteCallback	XmCCallback	XmRCallback
XmNxrtFldConvertCallback	XmCCallback	XmRCallback
XmNxrtFldDefaultBackground	XmCXrtFldBackground	XmRxrtFldPixel
XmNxrtFldDefaultForeground	XmCXrtFldForeground	XmRxrtFldPixel
XmNxrtFldDisplayFormat	XmCXrtFldFormat	XmRxrtFldFormat
XmNxrtFldDisplayHorizScrollBar	XmCXrtFldDispScrollBar	XmRxrtFldDispScrollBar
XmNxrtFldDisplayVertScrollBar	XmCXrtFldDispScrollBar	XmRxrtFldDispScrollBar
XmNxrtFldEditFormat	XmCXrtFldFormat	XmRxrtFldFormat

Name	Class	Type
XmNxrtFldEnterCallback	XmCCallback	XmRCallback
XmNxrtFldErrorBackground	XmCXrtFldBackground	XmRxrtFldPixel
XmNxrtFldErrorCallback	XmCCallback	XmRCallback
XmNxrtFldErrorForeground	XmCXrtFldForeground	XmRxrtFldPixel
XmNxrtFldErrorString	XmCXrtFldMessageString	XmRXmString
XmNxrtFldExitCallback	XmCCallback	XmRCallback
XmNxrtFldInvalidChars	XmCXrtFldInvalidChars	XmRString
XmNxrtFldIsDefault	XmCReadOnly	XmRBoolean
XmNxrtFldIsModified	XmCReadOnly	XmRBoolean
XmNxrtFldIsValid	XmCReadOnly	XmRBoolean
XmNxrtFldLastError	XmCReadOnly	XmRxrtFldError
XmNxrtFldMask	XmCXrtFldMask	XmRString
XmNxrtFldMaskChars	XmCXrtFldMask	XmRString
XmNxrtFldMatchMask	XmCXrtFldMatchMask	XmRxrtFldMatchMask
XmNxrtFldMatchPickList	XmCBoolean	XmRBoolean
XmNxrtFldMaxLineLength	XmCMaxLength	XmRInt
XmNxrtFldMaxLines	XmCMaxLength	XmRInt
XmNxrtFldMenuAlignment	XmCXrtFldMenuAlignment	XmRXrtFldMenuAlignment
XmNxrtFldMenuCallback	XmCCallback	XmRCallback
XmNxrtFldMenuList	XmCReadOnly	XmRWidget
XmNxrtFldOverstrike	XmCBoolean	XmRBoolean
XmNxrtFldPickListIndex	XmCXrtFldPickListIndex	XmRInt
XmNxrtFldPromptString	XmCXrtFldMessageString	XmRXmString
XmNxrtFldPopupType	XmCXrtFldPopupType	XmRXrtFldPopupType
XmNxrtFldRange	XmCXrtFldRange	XmRxrtFldRange
XmNxrtFldSearchPolicy	XmCXrtFldSearchPolicy	XmRxrtFldSearchPolicy
XmNxrtFldSpinCallback	XmCCallback	XmRCallback
XmNxrtFldValidateCallback	XmCCallback	XmRCallback
XmNxrtFldValidChars	XmCXrtFldValidChars	XmRString
XmNxrtFldVisibleItemCount	XmCVisibleItemCount	XmRInt

F.2 XrtStringField Resources

Name	Class	Type
XmNxrtFldDefaultValue	XmCValue	XmRString
XmNxrtFldInternalValue	XmCString	XmRString
XmNxrtFldMax	XmCMaximum	XmRString
XmNxrtFldMin	XmCMinimum	XmRString
XmNxrtFldPickList	XmCString	XmRStringTable

F.3 XrtIntField Resources

Name	Class	Type
XmNxrtFldCounterIncrement	XmCValue	XmRxrtFldInt
XmNxrtFldDefaultValue	XmCValue	XmRxrtFldInt
XmNxrtFldDisplayFormatString	XmCXrtFldDisplayFormatString	XmRString
XmNxrtFldInternalValue	XmCValue	XmRxrtFldInt
XmNxrtFldMax	XmCMaximum	XmRxrtFldInt
XmNxrtFldMin	XmCMinimum	XmRxrtFldInt
XmNxrtFldNegativeFormat	XmCXrtFldNegativeFormat	XmRxrtFldNegativeFormat
XmNxrtFldPickList	XmCXrtFldIntList	XmRxrtFldIntList
XmNxrtFldRange	XmCXrtFldRange	XmRxrtFldRange

F.4 XrtFloatField Resources

Name	Class	Type
XmNxrtFldCounterIncrement	XmCValue	XmRxrtFldFloat
XmNxrtFldDecimalPlaces	XmCValue	XmRxrtFldDecimalPlaces
XmNxrtFldDefaultValue	XmCValue	XmRxrtFldFloat
XmNxrtFldDisplayFormatString	XmCXrtFldDisplayFormatString	XmRString
XmNxrtFldInternalValue	XmCValue	XmRxrtFldFloat
XmNxrtFldLeadingZero	XmCBoolean	XmRBoolean
XmNxrtFldMax	XmCMaximum	XmRxrtFldFloat
XmNxrtFldMin	XmCMinimum	XmRxrtFldFloat
XmNxrtFldNegativeFormat	XmCXrtFldNegativeFormat	XmRxrtFldNegativeFormat
XmNxrtFldPickList	XmCXrtFldFloatList	XmRxrtFldFloatList
XmNxrtFldRange	XmCXrtFldRange	XmRxrtFldRange

F.5 XrtDateField Resources

Name	Class	Type
XmNxrtFldCounterIncrement	XmCValue	XmRxrtFldDateIncrement
XmNxrtFldDefaultValue	XmCValue	XmRxrtFldDate
XmNxrtFldDisplayFormatString	XmCXrtFldDateFormat	XmRString
XmNxrtFldEnterFormatList	XmCXrtFldEnterFormatList	XmRStringTable
XmNxrtFldInternalValue	XmCValue	XmRxrtFldDate
XmNxrtFldInternalValueLong	XmCValue	XmRxrtFldInt
XmNxrtFldMax	XmCMaximum	XmRxrtFldDate
XmNxrtFldMilleniumThreshold	XmCXrtFldMilleniumThreshold	XmRInt
XmNxrtFldMin	XmCMinimum	XmRxrtFldDate
XmNxrtFldPickList	XmCXrtFldDateList	XmRxrtFldDateList

F.6 XrtComboBox Resources

Name	Class	Type
XmNxrtComboArrow	XmCReadOnly	XmRWidget
XmNxrtComboArrowClass	XmCReadOnly	XmRPointer
XmNxrtComboArrowType	XmCXrtFldArrowType	XmRxrtFldArrowType
XmNxrtComboField	XmCReadOnly	XmRWidget
XmNxrtComboFieldClass	XmCReadOnly	XmRPointer
XmNxrtComboFieldType	XmCXrtFldFieldType	XmRxrtFldFieldType
XmNxrtComboSpacing	XmCXrtFldSpacing	XmRHorizontalDimension

F.7 XrtSpinBox Resources

Name	Class	Type
XmNxrtSpinArrowLayout	XmCXrtSpinArrowLayout	XmRxrtSpinArrowLayout
XmNxrtSpinArrowOrientation	XmCXrtSpinArrowOrientation	XmRxrtSpinArrowOrientation
XmNxrtSpinArrowType	XmCXrtFldArrowType	XmRxrtFldArrowType
XmNxrtSpinDecrArrow	XmCReadOnly	XmRWidget
XmNxrtSpinDecrArrowClass	XmCReadOnly	XmRPointer
XmNxrtSpinField	XmCReadOnly	XmRWidget
XmNxrtSpinFieldClass	XmCReadOnly	XmRPointer
XmNxrtSpinFieldName	XmCString	XmRString
XmNxrtSpinFieldType	XmCXrtFldFieldType	XmRxrtFldFieldType
XmNxrtSpinIncrArrow	XmCReadOnly	XmRWidget
XmNxrtSpinIncrArrowClass	XmCReadOnly	XmRPointer
XmNxrtSpinSpacing	XmCXrtFldSpacing	XmRHorizontalDimension
XmNxrtSpinType	XmCXrtSpinType	XmRxrtSpinType

F.8 XrtCalendar Resources

Name	Class	Type
XmNxrtCalendarCellHighlightColor	XmCBackground	XmRxrtFldPixel
XmNxrtCalendarCellHighlightThickness	XmCHighlightThickness	XmRDimension

XmNxrtCalendarCellMargin	XmCMargin	XmRDimension
XmNxrtCalendarCreateMonth	XmCBoolean	XmRBoolean
XmNxrtCalendarCreateTime	XmCBoolean	XmRBoolean
XmNxrtCalendarCreateYear	XmCBoolean	XmRBoolean
XmNxrtCalendarEnabled	XmCBoolean	XmRBoolean
XmNxrtCalendarFirstDayOfWeek	XmCXrtCalendarFirstDayOfWeek	XmRInt
XmNxrtCalendarFullWeekdayNames	XmCBoolean	XmRBoolean
XmNxrtCalendarHeaderBackground	XmCBackground	XmRxrtFldPixel
XmNxrtCalendarHeaderFontList	XmCFontList	XmRFontList
XmNxrtCalendarHeaderForeground	XmCForeground	XmRxrtFldPixel
XmNxrtCalendarHeaderNumChars	XmCXrtCalendarHeaderNumChars	XmRInt
XmNxrtCalendarMarginBottom	XmCMarginBottom	XmRDimension
XmNxrtCalendarMarginLeft	XmCMarginLeft	XmRDimension
XmNxrtCalendarMarginRight	XmCMarginRight	XmRDimension
XmNxrtCalendarMarginTop	XmCMarginTop	XmRDimension
XmNxrtCalendarMaxRollDate	XmCValue	XmRxrtFldDate
XmNxrtCalendarMinRollDate	XmCValue	XmRxrtFldDate
XmNxrtCalendarModifyCallback	XmCCallback	XmRCallback
XmNxrtCalendarSelectedBackground	XmCBackground	XmRxrtFldPixel
XmNxrtCalendarSelectedForeground	XmCForeground	XmRxrtFldPixel
XmNxrtCalendarSpecialDayBackground	XmCBackground	XmRxrtFldPixel
XmNxrtCalendarSpecialDayForeground	XmCForeground	XmRxrtFldPixel
XmNxrtCalendarSpecialDays	XmCXrtCalendarSpecialDays	XmRxrtCalendarSpecialDays
XmNxrtCalendarValue	XmCValue	XmRxrtFldDate
XmNxrtCalendarValueLong	XmCValue	XmRxrtFldInt

G

Resource File and GUI Builder Usage

Syntax
Example Resource File

This appendix lists syntax for setting XRT/field resources in resource files and when using GUI builder tools. It also lists a sample resource file.

XRT/field only defines resource converters for the data types not already defined by Xt and Motif.

All field resources can be set in a resource file except callback resources.

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 specifying a backslash (`\`) as 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 `XRTFLD_` or `XRTSPIN_` is dropped. The resource values can be entered in upper, lower, or mixed case. For

example, each of the following resource specifications causes the value XRTFLD_ARROWS_BEGINNING to be used for XrtFldArrowLayout:

```
*.XtXrtField.xrtFldArrowLayout: ARROWS_BEGINNING
*.XtXrtField.xrtFldArrowLayout: arrows_beginning
*.XtXrtField.xrtFldArrowLayout: ArRowS_BeginninG
```

Dates

Resources of type XmRxrtFldDate can be entered in any of the following formats:

```
YYYY-MM-DD hh:mm:ss
YYYY-MMM-DD hh:mm:ss
YYYY-MM-DD
YYYY-MMM-DD
YYYY-MMM-DD
hh:mm:ss
```

The strings "TODAY" (representing today's date as of midnight) and "NOW" (the current time) are also accepted.

Date Lists

Resources of type XmRxrtFldDateList are defined using a comma-separated list of dates (in any of the formats described above). For example:

```
*.DateField.xrtFldPickList: now,1980,1850-jan-01\
12:00,today,1955-3-10
```

Date Counter Increments

Resources of type XmRxrtFldDateIncrement are defined using the syntax *year-month-date hour:minute:second*. For example, 2-0-5 0:0:0 increments by two years and five days, and 0-0-0 0:30:0 increments by thirty minutes.

Floating-Point Numbers

Resources of type XmRxrtFldFloat accept any string that can be input to a float field, as well as the following:

MAXDOUBLE	the largest possible double-precision floating-point number
-MAXDOUBLE	the largest possible negative double-precision floating-point number

Floating-Point Lists

Resources of type XmRxrtFldFloatList accept a comma-separated list of floating-point numbers. For example:

```
*.FloatField.xrtFldPickList: -34,1.09,10000,2e5,(100)
```

Integers

Resources of type `XmRxrtFldInt` accept any value that can be input to an integer field, as well as the following:

`MAXLONG`	the largest number that can be stored in a variable defined to be of type *long*
`-MAXLONG`	the largest negative number that can be stored in a variable of type *long*
`MAXINT`	the largest number that can be stored in a variable defined to be of type *int*
`-MAXINT`	the largest negative number that can be stored in a variable of type *int*

Integer Lists

Resources of type `XmRxrtFldIntList` accept a comma-separated list of integers. For example:

```
*.IntField.xrtFldPickList: -34,34,0b11101,0xA112
```

Color Specification

Color resources (`XmRxrtFldPixel` type) are specified using any string that can be converted to a color, as well as the following:

`UNCHANGED_COLOR`	use the `XmNbackground` or `XmNforeground` color inherited from the Core widget class)
`ERROR_COLOR`	use the default foreground or background color for widgets containing invalid values

G.2 Example Resource File

The following resource file controls the behavior of the *sampler* demo, which is described in Appendix I on page 171:

```
! sampler.res - resource file for sampler demo
!! General
*.background: #D1C1AB
*.foreground: black
*.XmToggleButton.selectColor: Yellow
*.fontList: -*-helvetica-bold-r-normal--14-140-*-*-*-*-*-*
*.field.fontList: 9x15
*.XmXrtCurrencyField.background: #FFF7D4
*.XmXrtCurrencyField.foreground: black
*.XmXrtDateField.background: #FFF7D4
*.XmXrtDateField.foreground: black
*.XmXrtFloatField.background: #FFF7D4
*.XmXrtFloatField.foreground: black
*.XmXrtIntField.background: #FFF7D4
*.XmXrtIntField.foreground: black
*.XmXrtStringField.background: #FFF7D4
*.XmXrtStringField.foreground: black
*.XmList.foreground: black
*.XmList.background: #FFF7D4
*.selections.fontList: -*-new century schoolbook-medium-r-normal-*-140-*

! Don't display non-message panedWindow sashes
*.sashWidth: 0
*.sashHeight: 0
*.sashShadowThickness: 0
*.sash.traversalOn: false
*.msg_pane.sashWidth: 10
*.msg_pane.sashHeight: 10
*.msg_pane.sashShadowThickness: 5

!*.xrtFldDefaultBackground: yellow
!*.xrtFldDefaultForeground: black

*.XmLabel.alignment: alignment_end
*.prompt_label.labelString: \n \n \n
*.prompt_label.alignment: alignment_center
*.prompt_label.recomputeSize: false
*.prompt_frame.shadowThickness: 5
*.error_label.labelString: \n \n
*.error_label.foreground: red
*.error_label.alignment: alignment_center
*.error_label.recomputeSize: false
*.error_frame.shadowThickness: 5

*.Help.title: Field Help
*.Help.XmLabel.alignment: alignment_beginning
*.Help.*.labelString: \n
*.Help.*.width: 500
*.Help.*.height: 100
*.Help.x: 500
*.Help.y: 500
*.Help.*.fontList:
-*-new century schoolbook-medium-r-normal-*-140-*=roman,\
```

```
-*-new century schoolbook-medium-i-normal-*-140-*=italic,\
-*-new century schoolbook-bold-r-normal-*-140-*=bold,\
-*-symbol-medium-r-normal-*-140-*=symbol
*.aboutXRTfield.fontList:
-*-new century schoolbook-medium-r-normal-*-140-*=roman,\
-*-new century schoolbook-medium-i-normal-*-140-*=italic,\
-*-new century schoolbook-bold-r-normal-*-140-*=bold,\
-*-symbol-medium-r-normal-*-140-*=symbol

!! Edit and display formats
*.Edit_Format.labelString: Edit Format:
*.Display_Format.labelString: Display Format:
*.format_bb.*.xrtFldDefaultBackground: unchanged_color
*.format_bb.*.xrtFldMatchPickList: false
*.format_bb.*.columns: 25

!! Fields
*.File_field.value: sample
*.File_field.xrtFldComplete: true
*.File_field.translations: #override\n\
Ctrl <Key>c: FileComplete()
*.File_field.xrtFldInvalidChars: [ ]
*.File_field.xrtFldErrorString: Spaces and brackets are not allowed
*.File_field.xrtFldPromptString: Press @B@CTRL-c@P@ to complete the filename \
(add a @B@.c@P@ suffix)
*.File_field.fontList: 9x15

*.Phone1_field.xrtFldMask: (@@@) @@@-@@@@ Ext. @@@@
*.Phone1_field.xrtFldAllowNull: false
*.Phone1_field.xrtFldInternalValue: 4165551212
*.Phone1_field.xrtFldMatchMask: match_partial_l_to_r
*.Phone1_field.xrtFldErrorString: Only numbers may be entered
*.Phone1_field.xrtFldPromptString: Enter a phone number, with optional extension
*.Phone1_field.fontList: 9x15
*.Phone1_field.columns: 25

*.Phone2_field.*.xrtFldMask: (@@@) @@@-@@@@
*.Phone2_field.*.fontList: 9x15
*.Phone2_field.*.xrtFldInternalValue: 4165551212
*.Phone2_field.*.xrtFldErrorString: Only numbers may be entered
*.Phone2_field.*.xrtFldPromptString: Select an international or local phone number\
mask from the picklist.

*.Phone2_Mask_combo.*.xrtFldPickList: Local 1,Local 2,International
*.Phone2_Mask_combo.*.xrtFldPromptString: Select an international or local phone\
number mask from the picklist.
*.Phone2_Mask_combo.*.fontList: 9x15

*.Float_field.xrtFldErrorString: Only numbers may be entered
*.Float_field.xrtFldPromptString: Enter any number
*.Float_field.fontList: 9x15

*.Date_field.xrtFldMask: @@@@-@@-@@ @@:@@:@@
*.Date_field.value: yyyy-mm-dd hh:mm:ss
*.Date_field.xrtFldErrorString: Only numbers may be entered
*.Date_field.xrtFldPromptString: Enter a date/time
*.Date_field.fontList: 9x15

*.ZipCode.labelString: Zip Code:
*.ZipCode_field.xrtFldMask: @@@@@-@@@@
```

```
*.ZipCode_field.xrtFldDefaultValue: 000000000
*.ZipCode_field.xrtFldMatchMask: match_partial_l_to_r
*.ZipCode_field.xrtFldErrorString: Only 5 or 9 digits may be entered
*.ZipCode_field.xrtFldPromptString: Enter a 5 or 9-digit number
*.ZipCode_field.fontList: 9x15

*.PostalCode.labelString: Postal Code:
*.PostalCode_field.xrtFldMask: U@U @U@
*.PostalCode_field.xrtFldDefaultValue: A1A 1A1
*.PostalCode_field.xrtFldErrorString: Enter a letter or a number
*.PostalCode_field.xrtFldPromptString: Enter a 6-character code
*.PostalCode_field.fontList: 9x15

*.Name_field.xrtFldPromptString: A Convert callback uppercases the first \
letter\nin each word
*.Name_field.xrtFldDisplayHorizScrollBar: dispsb_as_needed
*.Name_field.fontList: 9x15
*.Name_field.height: 40

*.Password_field.xrtFldAllowNull: false
*.Password_field.xrtFldErrorString: A password must be entered
*.Password_field.xrtFldPromptString: Enter any string
*.Password_field.xrtFldDisplayHorizScrollBar: dispsb_as_needed
*.Password_field.fontList: 9x15

*.Comment1_field.editMode: multi_line_edit
*.Comment1_field.xrtFldMaxLineLength: 10
*.Comment1_field.rows: 2
*.Comment1_field.xrtFldMaxLines: 3
*.Comment1_field.value: A\nsample\ncomment
*.Comment1_field.xrtFldPromptString: Only 3 lines, max 10 characters/line,\n\
can be entered
*.Comment1_field.fontList: 9x15

!! Combo boxes
*.Date_combo.*.xrtFldMatchPickList: false
*.Date_combo.*.xrtFldPickList: now,1980,1850-jan-01 12:00,today,1955-mar-10 00,\
1988-jun-01,2001-02-27 15:00
*.Date_combo.*.xrtFldErrorString: A valid date must be entered

*.Salaries_combo.*.xrtFldPickList: 5000,1000,15000,20000
*.Salaries_combo.*.xrtFldMatchPickList: false
*.Salaries_combo.*.xrtFldPromptString: Try setting the Edit and Display Formats for\
this field.
*.Salaries_combo.*.xrtFldErrorString: A valid currency must be entered.

*.Names_combo.*.xrtFldPickList: John Jones,Harry Smith,Jean Jones,Mary Martin,\
Harry Hayes

*.Int_combo.*.xrtFldDisplayFormatString: Value: %7d
*.Int_combo.*.xrtFldPickList: -34,100,10000
*.Int_combo.*.xrtFldPromptString: This field uses an XmNxrtFldDisplayFormatString \
\n\to customize the appearance.\\nThe entered value must be one of the picklist\
items.
*.Int_combo.*.xrtFldErrorString: A valid integer must be entered which \
matches one of the picklist items.

*.Float_combo.*.xrtFldPickList: -34,1.09,10000,2e5
*.Float_combo.*.xrtFldPromptString: Try setting the Edit and Display Formats for \
this field.
```

```
!! Spin boxes
*.Zodiac_spin.*.xrtFldPickList: Aries,Taurus,Gemini,Cancer,Leo,Virgo,Libra,\
Scorpio,Sagittarius,Capricorn,Aquarius,Pisces
*.Zodiac_spin.xrtSpinArrowLayout: arrows_beginning
*.Titles_spin.*.xrtFldPickList: Mr.,Mrs.,Ms.,Miss,Dr.
*.Primes_spin.*.xrtFldPickList: 1,3,5,7,11,13,17,19,23,29,31
*.Primes_spin.xrtSpinArrowLayout: arrows_end

!! Counter boxes
*.Integer_counter.xrtSpinArrowLayout: arrows_end
*.Integer_counter.xrtSpinArrowOrientation: vertical
*.Integer_counter.*.xrtFldInternalValue: 0
*.Integer_counter.*.xrtFldMin: -10
*.Integer_counter.*.xrtFldMax: 10
*.Integer_counter.*.xrtFldPromptString: Try setting the Edit and Display Formats\
for this field.

*.Integer2_counter.xrtSpinArrowLayout: arrows_end
*.Integer2_counter.xrtSpinArrowOrientation: vertical
*.Integer2_counter.*.xrtFldMin: 0
*.Integer2_counter.*.xrtFldCounterIncrement: 10
*.Integer2_counter.*.xrtFldDisplayFormat: format_hex
*.Integer2_counter.*.xrtFldInternalValue: 1
*.Integer2_counter.*.xrtFldPromptString: Try setting the Edit and Display Formats\
for this field.

*.Float_counter.*.xrtFldDefaultValue: 1
*.Float_counter.xrtSpinArrowLayout: arrows_end
*.Float_counter.xrtSpinArrowOrientation: vertical
*.Float_counter.*.xrtFldPromptString: This demonstrates using a @B@Spin \
Callback@P@\nto change the value by a power of 10.\\nTry setting the Edit and \
Display Formats for this field.

*.Date_counter.xrtSpinArrowLayout: arrows_end
*.Date_counter.xrtSpinType: type_counter
*.Date_counter.*.xrtFldInternalValue: today
*.Date_counter.day.columns: 2
*.Date_counter.day.xrtFldCounterIncrement: 0-0-1
*.Date_counter.day.xrtFldDisplayFormatString: DD
*.Date_counter.day.xrtFldEnterFormatList: DD
*.Date_counter.day.xrtFldPromptString: Pressing the arrow buttons changes the\
@B@day@P@
*.Date_counter.month.columns: 10
*.Date_counter.month.xrtFldCounterIncrement: 0-1-0
*.Date_counter.month.xrtFldDisplayFormatString: MMMM
*.Date_counter.month.xrtFldEnterFormatList: MMMM
*.Date_counter.month.xrtFldPromptString: Pressing the arrow buttons changes the\
@B@month@P@
*.Date_counter.wday.columns: 10
*.Date_counter.wday.xrtFldCounterIncrement: 0-0-1
*.Date_counter.wday.xrtFldDisplayFormatString: WW
*.Date_counter.wday.xrtFldEnterFormatList: WW
*.Date_counter.wday.xrtFldPromptString: Pressing the arrow buttons changes the\
@B@weekday@P@
*.Date_counter.year.columns: 4
*.Date_counter.year.xrtFldCounterIncrement: 1-0-0
*.Date_counter.year.xrtFldDisplayFormatString: YYYY
*.Date_counter.year.xrtFldEnterFormatList: YYYY
*.Date_counter.year.xrtFldPromptString: Pressing the arrow buttons changes the\
```

```
@B@year@P@
*.Date_counter.*.fontList: 9x15

!! Currencies
*.currency_bb.*.xrtFldPickList: 0,1000,1000000,-100
*.currency_bb.*.xrtFldMatchPickList: false
*.currency_bb.*.xrtFldPromptString: This illustrates the use of\
@I@XrtFldSetDefaultLocaleStruct()@P@\\nto set the locale for individual fields.\
\nTry setting the Negative Format and number of decimal places for this field.
*.Negative_Format.labelString: Negative Format:
*.Decimal_Places.labelString: Decimal Places:

*.UK_combo.*.xrtFldPickList: 0,1000.00,1000000,-100
*.UK_Intl.labelString: UK Intl:
*.UK_Intl_combo.*.xrtFldPickList: 0,1000.50,1000000,-100
*.UK_Intl_combo.*.xrtFldDisplayFormat: format_currency_intl

*.US_combo.*.xrtFldPickList: 0,1000.50,1000000,-100
*.US_Intl_combo.*.xrtFldPickList: 0,1000.50,1000000,-100
*.US_Intl_combo.*.xrtFldDisplayFormat: format_currency_intl
*.US_Intl.labelString: US Intl:

*.Japan_combo.*.xrtFldPickList: 0,1000.50,1000000,-100
*.Japan_Intl_combo.*.xrtFldPickList: 0,1000,1000000,-100
*.Japan_Intl_combo.*.xrtFldDisplayFormat: format_currency_intl
*.Japan_Intl.labelString: Japan Intl:

*.Swiss_Intl_combo.*.xrtFldDisplayFormat: format_currency_intl
*.Swiss_combo.*.xrtFldPickList: 0,1000.50,1000000,-100
*.Swiss_Intl_combo.*.xrtFldPickList: 0,1000.50,1000000,-100
*.Swiss_Intl.labelString: Swiss Intl:

*.Decimal_Places_counter.xrtSpinArrowLayout: arrows_end
*.Decimal_Places_counter.xrtSpinArrowOrientation: vertical
*.Decimal_Places_counter.*.xrtFldMin: 0
*.Decimal_Places_counter.*.xrtFldMax: 10

! File Menu
*.File_cascade.mnemonic: F
*.Exit_button.mnemonic: X
*.Exit_button.accelerator: Ctrl<Key>X
*.Exit_button.acceleratorText: Ctrl+X

! Help Menu
*.Help_cascade.mnemonic: H

! Date Popup
*.XmXrtCalendar.xrtCalendarCreateTime: true
*.XmXrtCalendar.fontList: 9x15
*.XmXrtCalendar.xrtCalendarHeaderFontList: -*-helvetica-bold-r-normal--\
14-140-*-*-*-*-*-*
*.XmXrtCalendar.xrtCalendarSpecialDays: (SPECIAL_DAY_BY_DATE 25 0 0 11 \
XRTCALENDAR_ALL) (SPECIAL_DAY_BY_DATE 1 0 0 0 XRTCALENDAR_ALL) \
(SPECIAL_DAY_BY_DAY_OF_WEEK 0 0 XRTCALENDAR_ALL XRTCALENDAR_ALL XRTCALENDAR_ALL)
*.XmXrtCalendar.xrtCalendarSpecialDayBackground: #FFF7D4
```

H

UIL Arguments

Common XrtField Resources ■ *XrtIntField Resources*
XrtFloatField Resources ■ *XrtDateField Resources*
XrtComboBox Resources ■ *XrtSpinBox Resources*
XrtCalendar Resources ■ *Reasons*

This appendix provides an alphabetical listing of XRT/field's UIL arguments and their data types. Another table provides XRT/field's UIL reasons. For more information on using UIL with XRT/field, see section 4.17 on page 87.

H.1 Common XrtField Resources

UIL Argument Name	Argument Type
XmNxrtFldAllowNull	boolean
XmNxrtFldCase	integer
XmNxrtFldDisplayFormat	integer
XmNxrtFldDisplayFormatString	string
XmNxrtFldDisplayHorizScrollBar	integer
XmNxrtFldDisplayVertScrollBar	integer
XmNxrtFldEditFormat	integer
XmNxrtFldInvalidChars	string
XmNxrtFldIsDefault	boolean
XmNxrtFldIsModified	boolean
XmNxrtFldIsValid	boolean
XmNxrtFldLastError	integer
XmNxrtFldMask	string

UIL Argument Name	Argument Type
XmNxrtFldMaskChars	string
XmNxrtFldMatchMask	integer
XmNxrtFldMatchPickList	boolean
XmNxrtFldMaxLineLength	integer
XmNxrtFldOverstrike	boolean
XmNxrtFldPickListIndex	integer
XmNxrtFldRange	integer
XmNxrtFldValidChars	string
XmNxrtFldVisibleItemCount	integer

H.2 XrtIntField Resources

UIL Argument Name	Argument Type
XmNxrtFldCounterIncrement	integer

H.3 XrtFloatField Resources

UIL Argument Name	Argument Type
XmNxrtFldDecimalPlaces	integer
XmNxrtFldLeadingZero	boolean
XmNxrtFldNegativeFormat	integer

H.4 XrtDateField Resources

UIL Argument Name	Argument Type
XmNxrtFldInternalValueLong	integer

H.5 XrtComboBox Resources

UIL Argument Name	Argument Type
XmNxrtComboFieldType	integer

H.6 XrtSpinBox Resources

UIL Argument Name	Argument Type
XmNxrtSpinArrowLayout	integer
XmNxrtSpinFieldName	string
XmNxrtSpinFieldType	integer
XmNxrtSpinType	integer

H.7 XrtCalendar Resources

UIL Argument Name	Argument Type
XmNxrtCalendarCellMargin	integer
XmNxrtCalendarCellHighlightColor	color
XmNxrtCalendarCellHighlightThick	integer
XmNxrtCalendarCreateMonth	boolean
XmNxrtCalendarCreateTime	boolean
XmNxrtCalendarCreateYear	boolean
XmNxrtCalendarEnabled	boolean
XmNxrtCalendarFirstDayOfWeek	integer
XmNxrtCalendarFullWeekdayNames	boolean
XmNxrtCalendarHeaderNumChars	integer
XmNxrtCalendarHeaderBackground	color
XmNxrtCalendarHeaderFontList	font
XmNxrtCalendarHeaderForeground	color
XmNxrtCalendarMarginBottom	integer
XmNxrtCalendarMarginLeft	integer
XmNxrtCalendarMarginRight	integer
XmNxrtCalendarMarginTop	integer

UIL Argument Name	Argument Type
XmNxrtCalendarSpecialDayBackground	color
XmNxrtCalendarSpecialDayForeground	color
XmNxrtCalendarSelectedBackground	color
XmNxrtCalendarSelectedForeground	color
XmNxrtCalendarValueLong	integer

H.8 Reasons

Reasons
XmNxrtFldCompleteCallback
XmNxrtFldConvertCallback
XmNxrtFldEnterCallback
XmNxrtFldErrorCallback
XmNxrtFldExitCallback
XmNxrtFldMenuCallback
XmNxrtFldSpinCallback
XmNxrtFldValidateCallback

I

Sample Code

Examples
Demos

This appendix provides an overview of the sample code included with XRT/field. XRT/field provides two types of programming samples–examples and demos. Examples are short programs that illustrate *specific* features. Demos are more complete applications that illustrate *several features* working together. Most common XRT/field programming tasks are covered in the examples and demos.

I.1 Examples

The following example programs are located in *$XRTHOME/src/field/examples*:

calendar.c	Displays a simple XrtCalendar widget.
combo.c	Displays a simple XrtComboBox widget.
counter.c	Displays an XrtSpinBox that counts from 0 to 10.
datespin.c	Creates an XrtSpinBox with four counters as children, containing the day of the week, the day of the month, the month, and the year. If any of these counters is incremented or decremented, the others are adjusted accordingly. (For example, if the day of the month is increased, the day of the week is automatically increased.) This means that this program can be used as a perpetual calendar.
datespin2.c	Creates a more advanced version of the above example, particularly showing the spinning of subfields.
fields.c	Displays an example of each of the five XrtField subclasses.

password.c	Uses a field to obtain a user password. The password typed by the user is written to *stdout*.
phone.c	Displays a phone number field. The area-code parentheses and dash are automatically supplied by means of a mask.
simple.c	The simple example program discussed in Chapter 1.
spin.c	Displays a simple XrtSpinBox widget.

I.2 Demos

Demo applications are located in *$XRTHOME/src/field/demos*. There is a directory for each demo:

form	This demo shows how you can use XRT/field widgets to create an employee data entry form. Many different kinds of XRT/field widgets are used for various employee data. For example, a combo box is used for the employee's job title; date fields with masks provide the date of birth and start date; and a currency field is used to input the salary.
locale	This subdirectory contains the *printlocale* program, which displays locale information for a selected locale or the current locale.
sampler	This demo shows all the features and capabilities of XRT/field. Separate windows (accessible via radio buttons) display collections of fields, spin boxes, combo boxes, counter boxes, and currencies. When an XRT/field widget gains input focus, the message window at the bottom of the application describes the capabilities of that widget.
table_entry	This demo shows how you can use XRT/field widgets with XRT/table. A simple table uses different field subclasses for editing different columns. For example, an integer field edits the Phone Number column. As a multiproduct demo, this is located in *$XRTHOME/src/common_demos*.

Index

A

action routines 142
  calendar() 142
  Cancel() 142
  Complete() 142
  Menu() 142
  NoAction() 142
  Overstrike() 142
  Scrollbars() 143
  Spin() 143
allowing empty fields 60
ANSI C, compatibility with XRT/field 12
attempting to traverse 26, 56, 106, 107
authorizing
  shared libraries 12
authorizing a program 11

B

binary notation 54
binding actions to keys 76

C

C++
  compatibility with XRT/field 12
  header file 87
  use of 87
C++ classes
  XrtField 87
Calendar Appearance 73
Calendar Callbacks 76
calendar date editor
  displaying 76
Calendar() Action 76
Calendar() action 95, 142
callback
  and XmText callbacks 85
  calendar 76
  completion 52, 77, 96
  conversion 51, 97
  enter 25, 103
  error 49, 104
    and XmText callbacks 56
  exit 27, 56, 107
    alternatives to 56
  menu 78, 111
  spin 80, 114
  validation 48, 115
  XmText 87
Cancel() action 77, 142
case conversion 96
changing validation resources 40
character mask 42, 108
character validation 41
checking multiple fields 81
colors
  background 57
  data type 57
  default values 57, 60
  error 50, 57
  field 60
  foreground 57
Combo Box
  customized arrow 69
  highlight font and searching pick list 67
  menu alignment 68
  modifying the pick list 67
  spacing 69, 72
  text field 68
compiling and authorizing a program 11
Complete() action 77, 142
completion 24, 27, 52, 96
  as action 77
  example 53
controlling decimal places 37
controlling scrollbars 79, 84
convenience routines 129
conversion 24, 25, 27, 51, 96
  callbacks 51, 97
    example 52
  performed before validation 51
Creating a Popup Calendar 17
Creating an XrtCalendar Widget 17
creating read-only widgets 38
customized display format 54

D

data type listing 145
data types 46
date conversion 83
date editor
  displaying 76

DAY_TO_SEC macro 139
Daygrid Header 72
decimal places 98
default translations 141
default values 59, 99
  and empty fields 60
  checking for 60
  colors 60, 98
  creating 59
  default 60
  not displaying 60
demo programs, discussion of 172
display format 24, 100
  and errors 49
  customized 54
    example 54
  displaying date fields 55
  example 53, 54
displaying a popup date editor 76
displaying a popup menu 78
displaying date fields 55
displaying in binary notation 100, 102
displaying in hexadecimal notation 100, 102
displaying in octal notation 54, 100, 102
displaying negative values 112
drag and drop operation 24

E

edit format 24, 25, 36, 102
  and errors 49
  for XrtDateField widgets 38
editing process 24
  complete 28
empty fields 60
enter callbacks 25, 103
enter field action 24, 25
enter prompt 61
entering the widget 24
environment variables, XRTHOME 9
error callbacks 49, 104
  example 50
error handling 40, 49
  colors 50, 104
  error types 50
  example 50
  overriding errors 50
event table 141
example programs, discussion of 171
executable, compiling and authorizing 11
exit callbacks 56, 107
  alternatives to 56
exiting the widget 24

F

field
  storing values 23
field display colors 60
field length validation 41
field width validation 41
Fields in Resource Files 63
field-wide resources, summary 32
fonts
  highlighting and Combo Box picklist 67

G

gaining input focus 24
getting started with XRT/field 9
giving up focus 24
GUI builders 30

H

handling empty fields 60
hexadecimal notation 54
highlight font
  and pick list 67
HOUR_TO_SEC macro 139

I

include files 11
Index 173
internationalization 88
introduction
  typographical conventions 3
introduction to XRT/field widget 9
invalid characters 41, 108
invalid fields
  value of 41
invalid value handling 49

K

keys
  binding actions to 76
keystroke action 24, 25
  conversion 25
  validation 25

L

learning XRT/field 9, 18
libraries 11
line length 110
linking
  shared libraries 12
literals in validation masks 43
locales
  printing 90
  using 88

M

mask validation 42, 108
masks and overstriking 44, 109
maximum line length 110
maximum number of lines 110
maximum value 110
menu callbacks 78, 111
Menu() action 27, 78, 142
methods 129
MIN_TO_SEC macro 139
minimum value 112
monochrome X Servers, use of 29
MONTH_TO_SEC macro 139
moving the mouse pointer 24, 26
multi-line text 58, 110
  and Motif 1.2 bug 58

N

negative values 112
NoAction() action 142

O

octal notation 54
output format, see display format
overriding an error 50
Overstrike() action 79, 142
overstriking 79, 109, 112
  and validation masks 44

P

paste action 24, 25
Paste Action Processing 25
paste operation 24
pick list
  and highlight font 67
  searching 66
pick lists 46, 110, 112
pointer resources 29
pressing Return 26, 56, 106, 107
  simulating XmText behavior 56
printing locales 90
printlocale 90
procedures 129
programmatic change action 24, 27
  conversion 27
  validation 27

R

range of valid values 46
read-only widgets 38
resource conversion to String 87
resource files 30
  example 162
  syntax 159
  use of 159
resource types and classes 153
resources
  field-wide 32
  summary of 31
reverse resource converters 87

S

sample code 171
scrollbars 101, 102
  accessing 84
  controlling 79, 84
Scrollbars() action 79, 143
setting floating-point resources 29
shared libraries
  linking and authorizing 12
simple example 10
Special Days 73
specifying an enter prompt 61
specifying no default value 60
Specifying the Spin Policy 71
spin callbacks 80, 114
spin policy, specifying 71
Spin() action 80, 143
storing dates 82
storing the current time in a field 83
storing values in fields 23
subfield, spin policy 71
summary of resources 31

T

technical support 3
text
  multi-line 58
    and Motif 1.2 bug 58
time conversion 83
translations/actions, listing of 141
typing keys into a field 24
typographical conventions 3

U

UIL
  arguments 167
  header file 87
  reasons 170
  use of 87
user exit action 24, 26
  attempting to traverse 26, 56
  completion 27
  conversion 27
  display format 27
  exit callbacks 27

moving the mouse pointer 26
pressing Return 26, 56
simulating XmText behavior 56
validation 27
user interaction 21
default user interaction 21
user widget entry 24
using locales 88

V

valid characters 41, 116
Validating Pasted Values 46
validation 12, 24, 25, 27, 40
callbacks 48, 115
changing validation resources 40
checking for valid value 40
data type 46
error handling 49
field length 41
field width 41
invalid characters 41
mask 42
and overstriking 44
literals in 43
partial matching 44
special character replacement 85
visual cues 45
of initial value 27
pick list 46
range 46
valid characters 41
validating multiple fields 81
validation process, see editing process
value of invalid fields 41

W

widget
basic concepts 19
calendar 17
Motif class hierarchy 20
Motif integration 19
XrtCalendar 17
widget classes 19
widget exit 24

X

XmCreateXrtCalendar() 129
XmCreateXrtComboBox() 129
XmCreateXrtCurrencyField() 130
XmCreateXrtDateField() 130
XmCreateXrtField() 87
XmCreateXrtFloatField() 130
XmCreateXrtIntField() 130
XmCreateXrtSpinBox() 131
XmCreateXrtStringField() 131
XmIsXrtCalendar() 131
XmIsXrtComboBox() 131
XmIsXrtCurrencyField() 131
XmIsXrtDateField() 132
XmIsXrtField() 132
XmIsXrtFloatField() 132
XmIsXrtIntField() 132
XmIsXrtSpinBox() 132
XmIsXrtStringField() 132
XmMULTI_LINE_EDIT 41, 58
XmNactivateCallback 82, 86, 105, 117
XmNautoShowCursorPosition 117
XmNbackground 57, 60
XmNblinkRate 117
XmNcolumns 117
XmNcursorPosition 117
XmNcursorPositionVisible 38, 117
XmNeditable 38, 117
XmNeditMode 41, 58, 110, 117
XmNfocusCallback 86, 117
XmNfocusPolicy 26
XmNfontList 67, 117
XmNforeground 57, 60
XmNgainPrimaryCallback 117
XmNlosePrimaryCallback 117
XmNlosingFocusCallback 86, 105, 107, 117
XmNmarginHeight 118
XmNmarginWidth 118
XmNmaxLength 41, 110, 118
XmNmodifyVerifyCallback 86, 105, 118
XmNmotionVerifyCallback 86, 118
XmNpendingDelete 118
XmNresizeHeight 118
XmNresizeWidth 118
XmNrows 58, 118
XmNselectionArray 118
XmNselectionArrayCount 118
XmNselectThreshold 118
XmNsource 118
XmNstringDirection 66, 69, 70
XmNtopCharacter 118
XmNvalue 23, 27, 49, 52, 53, 60, 99, 112, 118, 137
example 23
XmNvalueChangedCallback 86, 105, 107, 119
XmNvalueWcs 119
XmNverifyBell 119
XmNwordWrap 41, 110, 119
XmNxrtCalendarCellHighlightThickness 125
XmNxrtCalendarCellMargin 125
XmNxrtCalendarCreateMonth 125
XmNxrtCalendarCreateTime 125
XmNxrtCalendarCreateYear 125
XmNxrtCalendarEnabled 125
XmNxrtCalendarFirstDayofWeek 125
XmNxrtCalendarFullDayNames 125
XmNxrtCalendarHeaderBackground 125
XmNxrtCalendarHeaderFontList 125
XmNxrtCalendarHeaderForeground 126

XmNxrtCalendarHeaderNumChars 125
XmNxrtCalendarMarginBottom 126
XmNxrtCalendarMarginLeft 126
XmNxrtCalendarMarginRight 126
XmNxrtCalendarMarginTop 126
XmNxrtCalendarMaxRollDate 126
XmNxrtCalendarMinRollDate 126
XmNxrtCalendarModifyCallback 125
XmNxrtCalendarSelectedBackground 127
XmNxrtCalendarSelectedForeground 127
XmNxrtCalendarSpecialDayBackground 127
XmNxrtCalendarSpecialDayForeground 127
XmNxrtCalendarSpecialDays 127
XmNxrtCalendarValue 127
XmNxrtCalendarValueLong 127
XmNxrtComboArrow 66, 121
XmNxrtComboArrowClass 66, 121
XmNxrtComboArrowType 69, 72, 121, 146
XmNxrtComboField 66, 68, 121
XmNxrtComboFieldClass 121
XmNxrtComboFieldType 15, 121, 149
XmNxrtComboSpacing 69, 121
XmNxrtFldAllowNull 95, 105
XmNxrtFldAutoComplete 95
XmNxrtFldCalendar 95
XmNxrtFldCalendarCallback 76, 95
XmNxrtFldCase 51, 96, 146
XmNxrtFldCompleteCallback 52, 96, 146
XmNxrtFldConvertCallback 51, 97, 147
XmNxrtFldCounterIncrement 71, 80, 83, 98
XmNxrtFldCreatePopups 65, 98
XmNxrtFldDecimalPlaces 37, 98, 100
XmNxrtFldDefaultBackground 57, 60, 98
XmNxrtFldDefaultForeground 57, 60, 98
XmNxrtFldDefaultValue 59, 99
XmNxrtFldDisplayFormat 38, 53, 54, 56, 100, 101, 149
XmNxrtFldDisplayFormatString 39, 54, 100, 103
  example 54
XmNxrtFldDisplayHorizScrollBar 79, 84, 101, 143, 147
XmNxrtFldDisplayVertScrollBar 79, 84, 102, 143, 147
XmNxrtFldEditFormat 36, 37, 53, 102, 103, 149
  not used by date fields 40
XmNxrtFldEnterCallback 61, 103, 147
XmNxrtFldEnterFormatList 37, 38, 39, 103
  default 39
XmNxrtFldErrorBackground 57, 104
XmNxrtFldErrorCallback 41, 49, 86, 104, 148
XmNxrtFldErrorForeground 57, 106
XmNxrtFldErrorString 50, 106
XmNxrtFldExitCallback 56, 86, 107, 148
XmNxrtFldInternalValue 10, 23, 27, 36, 49, 52, 53, 60, 61, 77, 107, 137, 142
  and mask literals 44
  example 23
XmNxrtFldInternalValueLong 23, 108
XmNxrtFldInvalidChars 41, 42, 105, 108, 116
XmNxrtFldIsDefault 60, 98, 108
XmNxrtFldIsModified 108
XmNxrtFldIsValid 40, 104, 106, 108
XmNxrtFldLastError 108
XmNxrtFldLeadingZero 108
XmNxrtFldMask 42, 47, 64, 79, 105, 108
XmNxrtFldMaskChars 85, 109
XmNxrtFldMatchMask 44, 149
XmNxrtFldMatchPickList 46, 64, 110, 112
XmNxrtFldMax 12, 46, 71, 80, 105, 110, 113
XmNxrtFldMaxLineLength 41, 105, 110
XmNxrtFldMaxLines 41, 105, 110
XmNxrtFldMenuAlignment 68, 110
XmNxrtFldMenuCallback 79, 111, 149
XmNxrtFldMenuList 67, 111
XmNxrtFldMilleniumThreshold 111
XmNxrtFldMin 12, 46, 71, 80, 105, 112, 113
XmNxrtFldNegativeFormat 112, 150
XmNxrtFldOverstrike 44, 79, 109, 112, 142
XmNxrtFldPickList 14, 46, 64, 71, 77, 78, 80, 105, 112, 114, 142, 143
XmNxrtFldPickListIndex 64, 112
XmNxrtFldPopupType 113
XmNxrtFldPromptString 61, 103, 113
XmNxrtFldRange 110, 112, 113, 150
XmNxrtFldSearchPolicy 66, 113, 150
XmNxrtFldSpinCallback 114, 147, 151
XmNxrtFldSpinPolicy 71, 114
XmNxrtFldValidateCallback 48, 115, 133, 151
XmNxrtFldValidChars 41, 42, 105, 108, 116
XmNxrtFldVisibleItemCount 78, 116
XmNxrtGearSearchPolicy 67
XmNxrtPickListIndex 81
XmNxrtSpinArrowLayout 70, 122
XmNxrtSpinArrowOrientation 122
XmNxrtSpinArrowType 72, 122
XmNxrtSpinDecrArrow 72, 122
XmNxrtSpinDecrArrowClass 71, 122
XmNxrtSpinField 72, 122
XmNxrtSpinFieldClass 122
XmNxrtSpinFieldName 123
XmNxrtSpinFieldType 123, 149
XmNxrtSpinIncrArrow 72, 123
XmNxrtSpinIncrArrowClass 71, 123
XmNxrtSpinSpacing 72, 123
XmNxrtSpinType 71, 80, 123, 143
XmNxrtTblDataTypeContext 62
XmNXrtTblFieldArgList 63
XmNxrtTblFieldArgListContext 62, 64
XmNxrtTblStringPickListContext 64
XmRowColumn
  resources 120
XmSINGLE_LINE_EDIT 58
XmText 19, 20, 27, 56, 105, 107, 110, 141
  and XrtField callbacks 85
  parent of XrtField 20
  resources 117
  specifying callbacks 87
XmTextField
  and XRT/field 21
XmXrtComboBox, see XrtComboBox

XmXrtFldManager
parent of XrtComboBox 20
parent of XrtSpinBox 20
XmXrtSpinBox, see XrtSpinBox
XRT/field
and GUI builders 30
and XmText 20
and XmTextField 21
basic concepts 19
integration with Motif 19
libraries and include files 11
Motif class hierarchy 20
resource types and classes 153
resources 93
using resource files 159
validation 12
widget classes 19
XRT/field functionality 28
XRT/field, using with XRT/table 61
XRT/table
using XRT/field 61
xrt_auth program
flags 11
use of 11
XrtCalendar 19, 20
appearance 73
controlling display 142
creating 129
daygrid header 72
resource summary 35, 36
resource types and classes 157
special days 73
UIL resources 169
widget synopsis 94
XrtCalendar Resources 125
XrtCalendar widget 17
XrtCalendarAddSpecialDay() 132
XrtCalendarDeleteSpecialDay() 133
XrtCalendarDupSpecialDays() 133
XrtCalendarFreeSpecialDays() 133
XrtCalendarModifyCallbackStruct 145
XrtCalendarSpecialDay 145
XrtCalendarSpecialDayType 146
XrtCombo Box
searching the pick list 66
XrtComboBox 13, 19, 131
arrow class 66
arrow type 69, 72
arrow widget 15
child widget spacing 69, 72
creating 14, 129
default field settings 65
default popup menu resources 65
programming 65
resource summary 34
resource types and classes 157
retrieving child widget IDs 66
subclassed from XmXrtFldManager 20
UIL resources 169
user interaction 15
XrtCurrencyField
widget synopsis 93
XrtDateField
enter formats 103
resource summary 34
resource types and classes 156
UIL resources 168
widget synopsis 94
XrtField 19, 132, 141
and XmText callbacks 85
checking multiple fields 81
creating read-only widgets 38
default values 59
displaying a popup menu for 78
edit format 25
entering the widget 24
exit 24
functionality 28
keystrokes 24
replacing XmText widgets 20
resource types and classes 153
subclass of XmText 20
subclasses 21
UIL arguments 167
widget subclasses 21
XRTFLD_FIELDTYPE_STRING 15
XRTFLD_UNCHANGED_COLOR 60
XrtFldAction 146
XrtFldArrowType 146
XrtFldCalendarCallbackStruct 146
XrtFldCase 146
XrtFldCheckFields() 81, 133, 148
XrtFldCompleteCallbackStruct 146
XrtFldConvertCallbackStruct 147
XrtFldCvtLongToTime() 84, 133
XrtFldCvtStringToDate() 134
XrtFldCvtStringToTime() 83, 134
XrtFldCvtTimeToLong() 84, 134
XrtFldDirection 147
XrtFldDispScrollBar 147
XrtFldDoubleToArgVal() 29, 135
XrtFldDTconv 147
XrtFldEnterCallbackStruct 147
XrtFldError 148
XrtFldErrorCallbackStruct 148
XrtFldExitCallbackStruct 148
XrtFldFieldData 148
XrtFldFieldDataFree() 135
XrtFldFieldType 149
XrtFldFormat 149
XrtFldGetCurrencyLocale() 135
XrtFldGetDateLocale() 135
XrtFldGetDisplayValue() 136
XrtFldGetField() 136
XrtFldGetNumericLocale() 135
XrtFldGetPassword() 136
XrtFldMatchMask 149
XrtFldMenuAlignment 149
XrtFldMenuCallbackStruct 149
XrtFldMrmInitialize() 87, 136

XrtFldNegFormat 150
XrtFldPopupType 150
XrtFldRange 150
XrtFldReason 150
XrtFldSearchPolicy 150
XrtFldSetDefaultLocaleStruct() 90, 135, 137, 147
XrtFldSetReverseConverters() 88, 137
XrtFldSpinCallbackStruct 151
XrtFldSpinPolicy 151
XrtFldValidateCallbackStruct 151
XrtFldValidateValue() 137
XrtFloatField 36
  controlling decimal places 37
  decimal places 98
  resource summary 33
  resource types and classes 156
  UIL arguments 168
  widget synopsis 94
XRTHOME environment variable 9
XrtIntField 36, 53
  edit format 36
  overriding the edit format 36
  resource summary 33
  resource types and classes 155
  UIL arguments 168
  widget synopsis 94
XrtSpinArrowLayout 151
XrtSpinBox 14, 19, 132, 143, 151
  adding children 70
  and Spin action 80
  arrow class 71
  arrow location 70
  arrow orientation 70
  as counter 15
  children 15
  creating 15, 131
  default field settings 70
  obtaining child widget IDs 72
  programming 70
  resource summary 35
  resource types and classes 157
  spin data type 71
  subclassed from XmXrtFldManager 20
  UIL resources 169
XrtSpinType 151
XrtStringField
  resource summary 33
  resource types and classes 155
  widget synopsis 94
XrtTblEnableXrtField() 62
XrtTblGetXrtFieldByType() 63
XtVaTypedArg 57
XtXrt3d 20
XtXrtGraph 20
XtXrtTable 20

Y

YEAR_TO_SEC macro 139
yielding focus 24