Stardent

WINDOW SYSTEM TOOLKIT

Stardent Computer Inc.

Part Number: 340-0112-01

Change History

340-0035-02 Original
340-0112-01 January, 1990

CONTENTS

The *Window System Toolkit* manual contains:

- *X Toolkit Athena Widgets — C Language Interface*
- The following Intrinsics *man* pages:

ATHENA WIDGETS

CHAPTER ONE

X Toolkit Athena Widgets – C Language Interface

X Window System

X Version 11, Release 3

Ralph R. Swick

Digital Equipment Corporation
External Research Group
MIT Project Athena

Terry Weissman

Digital Equipment Corporation
Western Software Laboratory

Table of Contents

Acknowledgments

The implementation of the Athena Widgets was the responsibility of Ralph Swick, Ron Newman (Project Athena), and Mark Ackerman (Project Athena). Additional contributions to their implementation was made by:

Rich Hyde (Digital WSL)
Terry Weissman (Digital WSL)
Mary Larson (Digital UEG)
Joel McCormack (Digital WSL)
Jeanne Rich (Digital WSL)
Charles Haynes (Digital WSL)
Loretta Guarino-Reid (Digital WSL)

The contributors to the X10 toolkit also deserve much of the credit for this work. The Athena Widgets borrow heavily on the their counterparts in the X10 toolkit. The design and implementation of the X10 toolkit were done by:

Terry Weissman (Digital WSL)
Smokey Wallace (Digital WSL)
Phil Karlton (Digital WSL)
Charles Haynes (Digital WSL)
Ram Rao (Digital UEG)
Mary Larson (Digital UEG)
Mike Gancarz (Digital UEG)
Kathleen Langone (Digital UEG)

Thanks go to Al Mento of Digital's UEG Documentation Group for formatting and generally improving this document and to Chris Peterson of Project Athena for testing the many versions of the code and reviewing this document.

Ralph R. Swick
Digital Equipment Corporation
External Research Group
MIT Project Athena

Chapter 1

Athena Widgets and The Intrinsics

The Athena widget set and the Intrinsics make up the X Toolkit. In the X Toolkit, a widget is the combination of an X window or subwindow and its associated input and output semantics. The Athena widgets provide the base functionality necessary to build a wide variety of application environments. Because the Intrinsics mask implementation details from the widget and application programmer, the Athena widgets and the application environments built with them are fully compatible with the other widget sets built with the Intrinsics. For information about the Intrinsics, see the *X Toolkit Intrinsics – C Language Interface.*

The Athena widget set is a library package layered on top of the Intrinsics and Xlib. This layer extends the basic abstractions provided by X and provides the next layer of functionality primarily by supplying a cohesive set of sample widgets.

To the extent possible, the X Toolkit is policy free. The application environment, not the X Toolkit, defines, implements, and enforces:

- Policy
- Consistency
- Style

Each individual widget implementation defines its own policy. The X Toolkit design allows for but does not necessarily encourage the free mixing of radically differing widget implementations.

1.1. Introduction to the X Toolkit Library

The X Toolkit library provides tools that simplify the design of application user interfaces in the X Window System programming environment. It assists application programmers by providing a set of common underlying user-interface functions. It also lets widget programmers modify existing widgets or add new widgets. By using the X Toolkit library in their applications, programmers present a similar user interface across applications to all workstation users.

The X Toolkit consists of:

- A set of Intrinsics functions for building widgets
- An architectural model for constructing widgets
- A sample interface (widget set) for programming

While the majority of the Intrinsics functions are intended for the widget programmer, a subset of the Intrinsics functions are to be used by application programmers (see *X Toolkit Intrinsics – C Language Interface*). The architectural model lets the widget programmer design new widgets by using the Intrinsics and by combining other widgets. The application interface layers built on top of the X Toolkit include a coordinated set of widgets and composition policies. Some of these widgets and policies are specific to an application domain, and others are common across a number of application domains.

The X Toolkit also can implement one or more application interface layers to:

- Verify the toolkit architecture
- Provide a base set of widgets and composition policies that can be incorporated in other application interface layers
- Make the X Toolkit immediately usable by those application programmers who find that a supplied application interface layer meets their needs

The remainder of this chapter discusses the X Toolkit:

- Terminology
- Model
- Design principles and philosophy

1.2. Terminology

In addition to the terms already defined for X programming (see *Xlib – C Language X Interface*), the following terms are specific to the Intrinsics and used throughout this book.

Application programmer

A programmer who uses the X Toolkit to produce an application user interface.

Child

A widget that is contained within another ("parent") widget.

Class

The general group to which a specific object belongs.

Client

A function that uses a widget in an application or for composing other widgets.

Full name

The name of a widget instance appended to the full name of its parent.

Instance

A specific widget object as opposed to a general widget class.

Method

The functions or procedures that a widget class implements.

Name

The name that is specific to an instance of a widget for a given client.

Object

A software data abstraction consisting of private data and private and public functions that operate on the private data. Users of the abstraction can interact with the object only through calls to the object's public functions. In the X Toolkit, some of the object's public functions are called directly by the application, while others are called indirectly when the application calls the common Intrinsics functions. In general, if a function is common to all widgets, an application uses a single Intrinsics function to invoke the function for all types of widgets. If a function is unique to a single widget type, the widget exports the function as another "Xt" function.

Parent

A widget that contains at least one other ("child") widget. A parent widget is also known as a composite widget.

Resource

A named piece of data in a widget that can be set by a client, by an application, or by user defaults.

Superclass

A larger class of which a specific class is a member. All members of a class are also members of the superclass.

User

A person interacting with a workstation.

Widget

An object providing a user-interface abstraction (for example, a Scrollbar widget).

Widget class

The general group to which a specific widget belongs, otherwise known as the type of the widget.

Widget programmer

A programmer who adds new widgets to the X Toolkit.

1.3. Underlying Model

The underlying architectural model is based on the following premises:

Widgets are X windows

Every user-interface widget is contained in a unique X window. The X window ID for a widget is readily available from the widget ID, so standard Xlib window manipulation procedures can operate on widgets.

Information hiding

The data for every widget is private to the widget and its subclasses. That is, the data is neither directly accessible nor visible outside of the module implementing the widget. All program interaction with the widget is performed by a set of operations (methods) that are defined for the widget.

Widget semantics and widget layout geometry

Widget semantics are clearly separated from widget layout geometry. Widgets are concerned with implementing specific user-interface semantics. They have little control over issues such as their size or placement relative to other widget peers. Mechanisms are provided for associating geometric managers with widgets and for widgets to make suggestions about their own geometry.

1.4. Design Principles and Philosophy

The X Toolkit follows two design principles throughout, which cover languages and language bindings as well as widget IDs.

1.4.1. Languages and Language Bindings

The X Toolkit facilitates access from objective languages. However, the X Toolkit library is conveniently usable by application programs written in nonobjective languages. Procedural interface guidelines are required when the X Toolkit is used with nonobjective languages.

The guidelines for the procedural interfaces are:

- Strings are passed as null-terminated character arrays.
- Most other arrays are passed using two parameters: a size and a pointer to the first element.
- Most numeric arguments are passed by value.
- Structures as arguments are avoided, unless a method for building them is provided for languages without pointers. Pointers embedded in structures are allowed, but they should be avoided if an equivalent alternative is available.
- Pointers are not recommended as return arguments, unless they will never have to be dereferenced by the caller. If they need to be dereferenced, the caller should allocate storage and pass the address to the procedure to fill in.
- Procedures can be passed as parameters.
- The ownership of dynamically allocated storage is determined on a case-by-case basis. The application is also permitted to replace the standard memory allocation and freeing

routines used by the library at build time.

1.4.2. Widget IDs

All references to widgets use a unique identifier that is known as the widget ID. The widget ID is returned to the client by the **XtCreateWidget** function. From an application programmer's perspective, a widget ID is an opaque data type; no particular interpretation can be assigned to it. Given a widget ID, you can retrieve the corresponding X window ID, the **Display** and **Screen** structures, and other information by using Intrinsics functions.

From a widget programmer's perspective, the widget ID actually is a pointer to a data structure known as the widget instance record. Several parts of the data structure are common to all widget types, while other parts are unique to a particular widget type. The widget's private data that is associated with a particular widget instance normally is included directly in the widget instance record.

Chapter 2

Using Widgets

Widgets serve as the primary tools for building a user interface or application environment. The widget set consists of primitive widgets (for example, a command button) and composite widgets (for example, a Dialog widget).

The remaining chapters of this guide explain the widgets and the geometry managers that work together to provide a set of user-interface components. These user-interface components serve as a default interface for application programmers who do not want to implement their own widgets. In addition, they serve as examples or a starting point for those widget programmers who, using the Intrinsics mechanisms, want to implement alternative application programming interfaces.

This chapter discusses the common features of the X Toolkit widgets.

2.1. Initializing the Toolkit

You must invoke the toolkit initialization function **XtInitialize** before invoking any other toolkit routines. **XtInitialize** opens the X server connection, parses standard parts of the command line, and creates an initial widget that is to serve as the root of a tree of widgets that will be created by this application.

Widget XtInitialize(*shell_name*, *application_class*, *options*, *num_options*, *argc*, *argv*)
    String *shell_name*;
    String *application_class*;
    XrmOptionDescRec *options*[];
    Cardinal *num_options*;
    Cardinal **argc*;
    String *argv*[];

shell_name Specifies the name of the application shell widget instance, which usually is something generic like "main".

application_class
Specifies the class name of this application, which usually is the generic name for all instances of this application. By convention, the class name is formed by reversing the case of the application's first significant letter. For example, an application named "xterm" would have a class name of "XTerm".

options Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to **XrmParseCommand**. For further information, see *Xlib – C Language X Interface.*

num_options Specifies the number of entries in the options list.

argc Specifies a pointer to the number of command line parameters.

argv Specifies the command line parameters.

For further information about this function, see the *Intrinsics.*

2.2. Creating a Widget

Creating a widget is a three-step process. First, the widget instance is allocated, and various instance-specific attributes are set by using **XtCreateWidget**. Second, the widget's parent is informed of the new child by using **XtManageChild**. Finally, X windows are created for the parent and all its children by using **XtRealizeWidget** and specifying the top-most widget. The

first two steps can be combined by using **XtCreateManagedWidget**. In addition, **XtRealizeWidget** is automatically called when the child becomes managed if the parent is already realized.

To allocate and initialize a widget, use **XtCreateWidget**.

Widget XtCreateWidget(*name*, *widget_class*, *parent*, *args*, *num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

name	Specifies the instance name for the created widget that is used for retrieving widget resources.
widget_class	Specifies the widget class pointer for the created widget.
parent	Specifies the parent widget ID.
args	Specifies the argument list. The argument list is a variable-length list composed of name and value pairs that contain information pertaining to the specific widget instance being created. For further information, see Section 2.7.2.
num_args	Specifies the number of arguments in the argument list. When the num_args is zero, the argument list is never referenced.

When a widget instance is successfully created, the widget identifier is returned to the application. If an error is encountered, the **XtError** routine is invoked to inform the user of the error.

For further information, see the *Intrinsics*.

2.3. Common Arguments in the Widget Argument List

Although a widget can have unique arguments that it understands, all widgets have common arguments that provide some regularity of operation. The common arguments allow arbitrary widgets to be managed by higher-level components without regards to the individual widget type. All widgets ignore any argument that they do not understand.

The following resources are retrieved from the argument list or from the resource database by all X Toolkit widgets:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of the border in pixels
XtNdestroyCallback	XtCallbackList	NULL	Callback for **XtDestroyWidget**
XtNheight	Dimension	Widget dependent	Height of the widget
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNsensitive	Boolean	True	Whether widget should receive input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNwidth	Dimension	Widget dependent	Width of the widget
XtNx	Position	0	x coordinate within parent
XtNy	Position	0	y coordinate within parent

The following additional resources are retrieved from the argument list or from the resource database by many X Toolkit widgets:

Name	Type	Default	Description
XtNcallback	XtCallbackList	NULL	Callback functions and client data

Name	Type	Default	Description
XtNcursor	Cursor	None	Pointer cursor
XtNforeground	Pixel	XtDefaultForeground	Foreground color

The value for the XtNcursor resource can be specified in the resource database as a string, which can be specified as one of the following:

- A standard X cursor name from **<X11/cursorfont.h>**
- FONT font-name glyph-index [[font-name] glyph-index]
- A relative or absolute file name

The first font and glyph specify the cursor source pixmap. The second font and glyph specify the cursor mask pixmap. The mask font defaults to the source font, and the mask glyph index defaults to the source glyph index.

If a relative or absolute file name is specified, that file is used to create the source pixmap. Then the string "Mask" is appended to locate the cursor mask pixmap. If the "Mask" file does not exist, the suffix "msk" is tried. If "msk" fails, no cursor mask will be used. If a relative file name is used, the directory specified by the resource name **bitmapFilePath** or class **BitmapFilePath** is added to the beginning of the file name. If the **bitmapFilePath** resource is not defined, the default directory on a UNIX-based system is **/usr/include/X11/bitmaps**.

2.4. Realizing a Widget

The **XtRealizeWidget** function performs two tasks:

- Creates an X window for the widget and, if it is a composite widget, for each of its managed children.
- Maps each window onto the screen.

void XtRealizeWidget(*w*)
    Widget *w*;

w Specifies the widget.

For further information about this function, see the *X Toolkit Intrinsics – C Language Interface.*

2.5. Standard Widget Manipulation Functions

After a widget has been created, a client can interact with that widget by calling either of the following:

- One of the standard widget manipulation routines that provide functions that all widgets support
- A widget class-specific manipulation routine

The X Toolkit provides generic routines to provide the application programmer access to a set of standard widget functions. These routines let an application or composite widget manipulate widgets without requiring explicit knowledge of the widget type. The standard widget manipulation functions let you:

- Control the location, size and mapping of widget windows
- Destroy a widget instance
- Obtain an argument value
- Set an argument value

2.5.1. Mapping Widgets

By default, widget windows automatically are mapped (made viewable) by **XtRealizeWidget**. This behavior can be changed by using **XtSetMappedWhenManaged**, and it then is the client's responsibility to use the **XtMapWidget** function to make the widget viewable.

void XtSetMappedWhenManaged(*w*, *map_when_managed*)
Widget *w*;
Boolean *map_when_managed*;

w Specifies the widget.

map_when_managed
Specifies the new value. If map_when_managed is **True**, the widget is mapped automatically when it is realized. If map_when_managed is **False**, the client must call **XtMapWidget** or make a second call to **XtSetMappedWhen-Managed** to cause the child window to be mapped.

The definition for **XtMapWidget** is:

XtMapWidget(*w*)
Widget *w*;

w Specifies the widget.

When you create several children in sequence for a common parent after it has been realized, it is generally more efficient to construct a list of children as they are created and use **XtManageChildren** to inform their parent of them all at once, instead of causing each child to be managed separately. By managing a list of children at one time, the parent can avoid wasteful duplication of geometry processing and the associated "screen flash".

void XtManageChildren(*children*, *num_children*)
WidgetList *children*;
Cardinal *num_children*;

children Specifies a list of children to add.

num_children Specifies the number of children to add.

If the parent is already visible on the screen, it is especially important to batch updates so that the minimum amount of visible window reconfiguration is performed.

For further information about these functions, see the *Intrinsics*.

2.5.2. Destroying Widgets

To destroy a widget instance of any type, use **XtDestroyWidget**.

void XtDestroyWidget(*w*)
Widget *w*;

w Specifies the widget.

XtDestroyWidget destroys the widget and recursively destroys any children that it may have, including the windows created by its children. After calling **XtDestroyWidget**, no further references should be made to the widget or to the widget IDs of any children that the destroyed widget may have had.

2.5.3. Retrieving Widget Resource Values

To retrieve the current value of a resource attribute associated with a widget instance, use **XtGetValues**.

void XtGetValues(*w*, *args*, *num_args*)
Widget *w*;
ArgList *args*;
Cardinal *num_args*;

w Specifies the widget.

args Specifies a variable-length argument list of name and address pairs that contain the resource name and the address into which the resource value is stored.

num_args Specifies the number of arguments in the argument list.

The arguments and values passed in the argument list are dependent on the widget. Note that the caller is responsible for allocating space into which the returned resource value is copied; the **ArgList** contains a pointer to this storage. The caller must allocate storage of the type as represented in the widget. For example, x and y must be allocated as Position and so on. For further information, see the *X Toolkit Intrinsics – C Language Interface.*

2.5.4. Modifying Widget Resource Values

To modify the current value of a resource attribute associated with a widget instance, use **XtSetValues**.

void XtSetValues(*w*, *args*, *num_args*)
Widget *w*;
ArgList *args*;
Cardinal *num_args*;

w Specifies the widget.

args Specifies a variable-length argument list of name and value pairs that contain the arguments to be modified and their new values.

num_args Specifies the number of arguments in the argument list.

The arguments and values passed in the argument list depend on the widget being modified. Some widgets may not allow certain resources to be modified after the widget instance has been created or realized. No notification is given if any part of a **XtSetValues** request is ignored.

For further information about these functions, see the *Intrinsics*.

Note

The argument list entry for **XtGetValues** specifies the address to which the caller wants the value copied. The argument list entry for **XtSetValues**, however, contains the new value itself if the size of value is less than sizeof(XtArgVal) (architecture dependent, but at least sizeof(long)); otherwise, it is a pointer to the value. String resources are always passed as pointers, regardless of the length of the string.

2.6. Using the Client Callback Interface

Widgets communicate changes in their state to their clients by means of a callback facility. The format for a client's callback handler is:

void *CallbackProc*(*w*, *client_data*, *call_data*)
Widget *w*;
caddr_t *client_data*;
caddr_t *call_data*;

w Specifies widget for which the callback is registered.

client_data Specifies arbitrary client-supplied data that the widget should pass back to the client when the widget executes the client's callback procedure. This is a way for the client registering the callback to also register client-specific data: a pointer to additional information about the widget, a reason for invoking the callback, and so on. It is perfectly normal to have client_data of NULL if all necessary information is in the widget. This field is also frequently known as the *closure*.

call_data Specifies any callback-specific data the widget wants to pass to the client. For example, when Scrollbar executes its jumpProc callback list, it passes the current position of the thumb in the call_data argument.

Callbacks can be registered with widgets in one of two ways. When the widget is created, a pointer to a list of callback procedure and data pairs can be passed in the argument list to **XtCreateWidget**. The list is of type **XtCallbackList**:

```
typedef struct {
        XtCallbackProc callback;
        caddr_t closure;
} XtCallbackRec, *XtCallbackList;
```

The callback list must be allocated and initialized before calling **XtCreateWidget**. The end of the list is identified by an entry containing NULL in callback and closure. Once the widget is created, the client can change or de-allocate this list; The widget itself makes no further reference to it. The closure field contains the client_data passed to the callback when the callback list is executed.

The second method for registering callbacks is to use **XtAddCallback** after the widget has been created.

void XtAddCallback(*w*, *callback_name*, *callback*, *client_data*)
Widget *w*;
String *callback_name*;
XtCallbackProc *callback*;
caddr_t *client_data*;

w Specifies the widget to add the callback to.

callback_name Specifies the callback list within the widget to append to.

callback Specifies the callback procedure to add.

client_data Specifies the data to be passed to the callback when it is invoked.

XtAddCallback adds the specified callback to the list for the named widget.

All widgets provide a callback list named **XtNdestroyCallback** where clients can register procedures that are to be executed when the widget is destroyed. The destroy callbacks are executed when the widget or an ancestor is destroyed. The call_data argument is unused for destroy callbacks.

The X Toolkit Intrinsics provide additional functions for further manipulating a callback list. For information about these functions, see **XtCallCallbacks**, **XtRemoveCallback**,

XtRemoveCallbacks, and **XtRemoveAllCallbacks** in the *X Toolkit Intrinsics – C Language Interface.*

2.7. Programming Considerations

This section provides some guidelines to set up an application program that uses the X Toolkit. This section discusses:

- Writing applications
- Creating argument lists

2.7.1. Writing Applications

When writing an application that uses the toolkit, you should make sure that your application performs the following:

1. Include **<X11/Intrinsic.h>** in your application programs. This header file automatically includes **<X11/Xlib.h>**, so all Xlib functions also are defined.
2. Include the widget-specific header files for each widget type that you need to use. For example, **<X11/Label.h>** and **<X11/Command.h>**.
3. Call the **XtInitialize** function before invoking any other toolkit or Xlib functions. For further information, see Section 2.1 and the *X Toolkit Intrinsics – C Language Interface.*
4. To pass attributes to the widget creation routines that will over-ride any site or user customizations, set up argument lists. In this document, a list of valid argument names that start with **XtN** is provided in the discussion of each widget.

   For further information, see Section 2.7.2.
5. When the argument list is set up, create the widget by using the **XtCreateWidget** function. For further information, see Section 2.2 and the *X Toolkit Intrinsics – C Language Interface.*
6. If the widget has any callback routines, which are usually defined by the **XtNcallback** argument or the **XtAddCallback** function, declare these routines within the application.
7. After a widget has been created, use **XtManageChild** to manage it. If there is no manipulation of the widget between **XtCreateWidget** and **XtManageChild,** you can do this in a single step by using **XtCreateManagedWidget**. For further information about these functions, see the *Intrinsics.*
8. After creating the initial widget hierarchy, windows must be created for each widget by calling **XtRealizeWidget** on the top level widget.
9. Most applications now sit in a loop processing events using **XtMainLoop**, for example:

   XtCreateManagedWidget(*name, class, parent, args, num_args*);
   XtRealizeWidget(*parent*);
   XtMainLoop( );

   For information about this function, see the *X Toolkit Intrinsics – C Language Interface.*
10. Link your application with **libXaw.a** (the Athena widgets), **libXmu.a** (miscellaneous utilities), **libXt.a** (the X Toolkit Intrinsics), and **libX11.a** (the core X library). The following provides a sample command line:

    cc -o *application application*.c –lXaw –lXmu –lXt –lX11

2.7.2. Creating Argument Lists

To set up an argument list for the inline specification of widget attributes, you can use one of the four approaches discussed in this section. You should use whichever approach fits the needs of the application and you are most comfortable with. In general, argument lists should be kept as short as possible to allow widget attributes to be specified through the resource database. Whenever a client inserts a specific attribute value in an argument list, the user is prevented from customizing the behavior of the widget. Resource names in the resource database, by convention, correspond to their symbolic names that are used in argument list without the **XtN** prefix. For example, the resource name for **XtNforeground** is "foreground". For further information, see the *Intrinsics*.

The **Arg** structure contains:

```
typedef struct {
        String name;
        XtArgVal value;
} Arg, *ArgList;
```

The first approach lets you statically initialize the argument list. For example:

```
static Arg arglist[] = {
        {XtNwidth, (XtArgVal) 400},
        {XtNheight, (XtArgVal) 300},
};
```

This approach makes it easy to add or delete new elements. The **XtNumber** macro can be used to compute the number of elements in the argument list, thus preventing simple programming errors. The following provides an example:

```
XtCreateWidget(name, class, parent, arglist, XtNumber(arglist));
```

The second approach lets you use the **XtSetArg** macro. For example:

```
Arg arglist[10];
XtSetArg(arglist[1], XtNwidth, 400);
XtSetArg(arglist[2], XtNheight, 300);
```

To make it easier to insert and delete entries, you also can use a variable index, as in this example:

```
Arg arglist[10];
Cardinal i=0;
XtSetArg(arglist[i], XtNwidth,  400);      i++;
XtSetArg(arglist[i], XtNheight, 300);      i++;
```

The i variable can then be used as the argument list count in the widget create function. In this example, **XtNumber** would return 10, not 2, and therefore is not useful.

Note

You should not use auto-increment or auto-decrement within the first argument to **XtSetArg**. As it is currently implemented, **XtSetArg** is a macro that dereferences the first argument twice.

The third approach lets you individually set the elements of the argument list array, one piece at a time. For example:

```
Arg arglist[10];
arglist[0].name  = XtNwidth;
arglist[0].value = (XtArgVal) 400;
arglist[1].name  = XtNheight;
arglist[1].value = (XtArgVal) 300;
```

Note that in this example, as in the previous example, **XtNumber** would return 10, not 2, and therefore is not useful.

The fourth approach lets you use a mixture of the first and third approaches: you can statically define the argument list but modify some entries at runtime. For example:

```
static Arg arglist[] = {
        {XtNwidth, (XtArgVal) 400},
        {XtNheight, (XtArgVal) NULL},
};
arglist[1].value = (XtArgVal) 300;
```

In this example, **XtNumber** can be used, as in the first approach, for easier code maintenance.

2.7.3. Sample Program

The following program creates one command button that, when pressed, causes the program to exit. This example is a complete program that illustrates:

- Toolkit initialization
- Optional command-line arguments
- Widget creation
- Callback routines

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Command.h>

static XrmOptionDescRec options[] = {
{"-label", "*button.label", XrmoptionSepArg, NULL}
};

Syntax(call)
        char *call;
{
        fprintf(stderr, "Usage: %s\n", call);
}

void Activate(w, client_data, call_data)
        Widget w;
        caddr_t client_data;        /* unused */
        caddr_t call_data;          /* unused */
{
        printf("button was activated.\n");
        exit(0);
}

void main(argc, argv)
        unsigned int argc;
        char **argv;
{
        Widget toplevel;
        static XtCallbackRec callbacks[] = {
          { Activate, NULL },
          { NULL, NULL },
        };

        static Arg args[] = {
         { XtNcallback, (XtArgVal)callbacks },
        };

        toplevel = XtInitialize("main", "Demo", options, XtNumber(options), &argc, argv );
        if (argc != 1) Syntax(argv[0]);

        XtCreateManagedWidget("button",commandWidgetClass,toplevel,args,XtNumber(args));

        XtRealizeWidget(toplevel);
        XtMainLoop( );
}
```

Chapter 3

Athena Widget Set

This chapter describes the following Athena widgets:

- Command
- Label
- Text
- Scrollbar
- Viewport
- Box
- VPaned
- Form
- Dialog
- List
- Grip
- Toggle

3.1. Command Widget

The Command widget is a rectangular button that contains a text or pixmap label. When the pointer cursor is on the button, the button border is highlighted to indicate that the button is available for selection. Then, when a pointer button is pressed and released the button is selected, and the application's callback routine is invoked.

The class variable for the Command widget is **commandWidgetClass**.

When creating a Command widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNbitmap	Pixmap	None	Pixmap to display in place of the label
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of button border
XtNcallback	XtCallbackList	NULL	Callback for button select
XtNcursor	Cursor	None	Pointer cursor
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNfont	XFontStruct*	XtDefaultFont	Label font
XtNforeground	Pixel	XtDefaultForeground	Foreground color
XtNheight	Dimension	Text height	Button height
XtNhighlightThickness	Dimension	2	Width of border to be highlighted
XtNinsensitiveBorder	Pixmap	Gray	Border when not sensitive
XtNinternalHeight	Dimension	2	Internal border height for highlighting
XtNinternalWidth			

Name	Type	Default	Description
	Dimension		

Name	Type	Default	Description
			Internal border width for highlighting

Name	Type	Default	Description
XtNjustify	XtJustify	XtJustifyCenter	Type of text alignment
XtNlabel	String	Button name	Button label
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNresize	Boolean	True	Whether to auto-resize in SetValues
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	see below	Event-to-action translations
XtNwidth	Dimension	Text width	Button width
XtNx	Position	0	x coordinate
XtNy	Position	0	y coordinate

The new resources associated with the Command widget are:

XtNbitmap Specifies a bitmap to display in place of the text label. See the description of this resource in the Label widget for further details.

XtNheight Specifies the height of the Command widget. The default value is the minimum height that will contain:
XtNinternalheight + height of XtNlabel + XtNinternalHeight
If the specified height is larger than the minimum, the label string is centered vertically.

XtNinternalHeight Represents the distance in pixels between the top and bottom of the label text or bitmap and the horizontal edges of the Command widget. HighlightThickness can be larger or smaller than this value.

XtNinternalWidth Represents the distance in pixels between the ends of the label text or bitmap and the vertical edges of the Command widget. HighlightThickness can be larger or smaller than this value.

XtNjustify Specifies left, center, or right alignment of the label string within the Command widget. If it is specified within an **ArgList**, one of the values **XtJustifyLeft**, **XtJustifyCenter**, or **XtJustifyRight** can be specified. In a resource of type "string", one of the values "left", "center", or "right" can be specified.

XtNlabel Specifies the text string that is to be displayed in the Command widget if no bitmap is specified. The default is the widget name of the Command widget.

XtNresize Specifies whether the Command widget should attempt to resize to its preferred dimensions whenever **XtSetValues** is called for it. The default is **True**.

XtNsensitive If set to **False**, the Command widget will change its window border to **XtNinsensitiveBorder** and will stipple the label string.

XtNwidth Specifies the width of the Command widget. The default value

is the minimum width that will contain:
XtNinternalWidth + width of XtNlabel + XtNinternalWidth
If the width is larger or smaller than the minimum, **XtNjustify** determines how the label string is aligned.

The Command widget supports the following actions:

- Switching the button between the foreground and background colors with **set** and **unset**
- Processing application callbacks with **notify**
- Switching the internal border between highlighted and unhighlighted states with **highlight** and **unhighlight**

The following are the default translation bindings that are used by the Command widget:

```
<EnterWindow>:      highlight()
<LeaveWindow>:      reset()
<Btn1Down>:         set()
<Btn1Up>:           notify() unset()
```

With these bindings, the user can cancel the action before releasing the button by moving the pointer out of the Command widget.

3.1.1. Command Actions

The full list of actions supported by Command is:

highlight(*condition*) Displays the internal highlight border in the color (**XtNforeground** or **XtNbackground**) that contrasts with the interior color of the Command widget. This action procedure takes one of the following conditions: WhenUnset and Always. If no argument is passed then WhenUnset is assumed, this maintains backwards compatibility.

unhighlight() Displays the internal highlight border in the color (**XtNforeground** or **XtNbackground**) that matches the interior color of the Command widget.

set() Enters the "set" state, in which **notify** is possible and displays the interior of the button in the **XtNforeground** color. The label is displayed in the **XtNbackground** color.

unset() Cancels the "set" state and displays the interior of the button in the **XtNbackground** color. The label is displayed in the **XtNforeground** color.

reset() Cancels any **set** or **highlight** and displays the interior of the button in the **XtNbackground** color, with the label displayed in the **XtNforeground** color.

notify() Executes the **XtNcallback** callback list if executed in the **set** state. The value of the call_data argument is undefined.

To create a Command widget instance, use **XtCreateWidget** and specify the class variable **commandWidgetClass**.

To destroy a Command widget instance, use **XtDestroyWidget** and specify the widget ID of the button.

The Command widget supports two callback lists: **XtNdestroyCallback** and **XtNcallback**. The notify action executes the callbacks on the **XtNcallback** list. The call_data argument is unused.

3.2. Label Widget

A Label is an noneditable text string or pixmap that is displayed within a window. The string is limited to one line and can be aligned to the left, right, or center of its window. A Label can neither be selected nor directly edited by the user.

The class variable for the Label widget is **labelWidgetClass**.

When creating a Label widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNbitmap	Pixmap	None	Pixmap to display in place of the label
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Border width in pixels
XtNcursor	Cursor	None	Pointer cursor
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNfont	XFontStruct*	XtDefaultFont	Label font
XtNforeground	Pixel	XtDefaultForeground	Foreground color
XtNheight	Dimension	text height	Height of widget
XtNinsensitiveBorder	Pixmap	Gray	Border when not sensitive
XtNinternalHeight	Dimension	2	See note
XtNinternalWidth	Dimension	4	See note
XtNjustify	XtJustify	XtJustifyCenter	Type of text alignment
XtNlabel	String	label name	String to be displayed
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNresize	Boolean	True	Whether to auto-resize in SetValues
XtNsensitive	Boolean	True	Whether widget receives input
XtNwidth	Dimension	text width	Width of widget
XtNx	Position	0	x coordinate in pixels
XtNy	Position	0	y coordinate in pixels

The new resources associated with Label are:

XtNbitmap Specifies a bitmap to display in place of the text label. The bitmap can be specified as a string in the resource data base. The StringToPixmap converter will interpret the string as the name of a file in the bitmap utility format that is to be loaded into a pixmap. The string can be an absolute or a relative file name. If a relative file name is used, the directory specified by the resource name **bitmapFilePath** or the resource class **BitmapFilePath** is add to the beginning of the specified file name. If the **bitmapFilePath** resource is not defined, the default directory on a UNIX-based system is **/usr/include/X11/bitmaps**.

XtNheight Specifies the height of the Label widget. The default value is the minimum height that will contain:
XtNinternalheight + height of XtNlabel + XtNinternalHeight
If the specified height is larger than the minimum, the label string is centered vertically.

XtNinternalHeight	Represents the distance in pixels between the top and bottom of the label text or bitmap and the horizontal edges of the Label widget.
XtNinternalWidth	Represents the distance in pixels between the ends of the label text or bitmap and the vertical edges of the Label widget.
XtNjustify	Specifies left, center, or right alignment of the label string within the Label widget. If it is specified within an **ArgList**, one of the values **XtJustifyLeft**, **XtJustifyCenter**, or **XtJustifyRight** can be specified. In a resource of type "string", one of the values "left", "center", or "right" can be specified.
XtNlabel	Specifies the text string that is to be displayed in the button if no bitmap is specified. The default is the widget name of the Label widget.
XtNresize	Specifies whether the Label widget should attempt to resize to its preferred dimensions whenever **XtSetValues** is called for it.
XtNsensitive	If set to **False**, the Label widget will change its window border to **XtNinsensitiveBorder** and will stipple the label string.
XtNwidth	Specifies the width of the Label widget. The default value is the minimum width that will contain: XtNinternalWidth + width of XtNlabel + XtNinternalWidth If the width is larger or smaller than the minimum, **XtNjustify** determines how the label string is aligned.

To create a Label widget instance, use **XtCreateWidget** and specify the class variable **labelWidgetClass**.

To destroy a Label widget instance, use **XtDestroyWidget** and specify the widget ID of the label.

The Label widget supports only the **XtNdestroyCallback** callback list.

3.3. Text Widget

A Text widget is a window that provides a way for an application to display one or more lines of text. The displayed text can reside in a file on disk or in a string in memory. An option also lets an application display a vertical Scrollbar in the Text window, letting the user scroll through the displayed text. Other options allow an application to let the user modify the text in the window.

The Text widget is divided into three parts:

- Source
- Sink
- Text widget

The idea is to separate the storage of the text (source) from the painting of the text (sink). The Text widget coordinates the sources and sinks. Clients usually will use **AsciiText** widgets that automatically create the source and sink for the client. A client can, if it so chooses, explicitly create the source and sink before creating the Text widget.

The source stores and manipulates the text. The X Toolkit provides string and disk file sources. The source determines what editing functions may be performed on the text.

The sink obtains the fonts and the colors in which to paint the text. The sink also computes what text can fit on each line. The X Toolkit provides a single-font, single-color ASCII sink.

If a disk file is used to display the text, two edit modes are available:

- Append
- Read-only

Append mode lets the user enter text into the window, while read-only mode does not. Text may only be entered if the insertion point is after the last character in the window.

If a string in memory is used, the application must allocate the amount of space needed. If a string in memory is used to display text, three types of edit mode are available:

- Append-only
- Read-only
- Editable

The first two modes are the same as displaying text from a disk file. Editable mode lets the user place the cursor anywhere in the text and modify the text at that position. The text cursor position can be modified by using the key strokes or pointer buttons defined by the event bindings. Many standard keyboard editing facilities are supported by the event bindings. The following actions are supported:

Cursor Movement
- forward-character
- backward-character
- forward-word
- backward-word
- forward-paragraph
- backward-paragraph
- beginning-of-line
- end-of-line
- next-line
- previous-line
- next-page
- previous-page
- beginning-of-file
- end-of-file
- scroll-one-line-up
- scroll-one-line-down

Delete
- delete-next-character
- delete-previous-character
- delete-next-word
- delete-previous-word
- delete-selection

Selection
- select-word
- select-all
- select-start
- select-adjust
- select-end
- extend-start
- extend-adjust
- extend-end

New Line
- newline-and-indent
- newline-and-backup
- newline

Miscellaneous
- redraw-display
- insert-file
- insert-char
- insert-string
- do-nothing

Kill
- kill-word
- backward-kill-word

Unkill
- unkill
- stuff

kill-selection
kill-to-end-of-line
kill-to-end-of-paragraph

insert-selection

Note

1. A page corresponds to the size of the Text window. For example, if the Text window is 50 lines in length, scrolling forward one page is the same as scrolling forward 50 lines.
2. The **insert-char** action may only be attached to a key event. It calls XLookupString to translate the event into a (rebindable) Latin-1 character (sequence) and inserts that sequence into the text at the current position. The **insert-string** action takes one or more arguments and inserts the arguments into the text at the current position. An argument beginning with the characters "0x" and containing only valid hexadecimal digits in the remainder is interpreted as a hexadecimal constant and the corresponding single character is inserted instead.
3. The **delete** action deletes a text item. The **kill** action deletes a text item and puts the item in the kill buffer (X cut buffer 1).
4. The **unkill** action inserts the contents of the kill buffer into the text at the current position. The **stuff** action inserts the contents of the paste buffer (X cut buffer 0) into the text at the current position. The **insert-selection** action retrieves the value of a specified X selection or cut buffer, with fall-back to alternative selections or cut buffers.

The default event bindings for the Text widget are:

```
char defaultTextTranslations[] = "\
        Ctrl<Key>F:             forward-character() \n\
        Ctrl<Key>B:             backward-character() \n\
        Ctrl<Key>D:             delete-next-character() \n\
        Ctrl<Key>A:             beginning-of-line() \n\
        Ctrl<Key>E:             end-of-line() \n\
        Ctrl<Key>H:             delete-previous-character() \n\
        Ctrl<Key>J:             newline-and-indent() \n\
        Ctrl<Key>K:             kill-to-end-of-line() \n\
        Ctrl<Key>L:             redraw-display() \n\
        Ctrl<Key>M:             newline() \n\
        Ctrl<Key>N:             next-line() \n\
        Ctrl<Key>O:             newline-and-backup() \n\
        Ctrl<Key>P:             previous-line() \n\
        Ctrl<Key>V:             next-page() \n\
        Ctrl<Key>W:             kill-selection() \n\
        Ctrl<Key>Y:             unkill() \n\
        Ctrl<Key>Z:             scroll-one-line-up() \n\
        Meta<Key>F:             forward-word() \n\
        Meta<Key>B:             backward-word() \n\
        Meta<Key>I:             insert-file() \n\
        Meta<Key>K:             kill-to-end-of-paragraph() \n\
        Meta<Key>V:             previous-page() \n\
        Meta<Key>Y:             stuff() \n\
```

```
Meta<Key>Z:                    scroll-one-line-down() \n\
  :Meta<Key>d:                 delete-next-word() \n\
  :Meta<Key>D:                 kill-word() \n\
  :Meta<Key>h:                 delete-previous-word() \n\
  :Meta<Key>H:                 backward-kill-word() \n\
  :Meta<Key>\<:                beginning-of-file() \n\
  :Meta<Key>\>:                end-of-file() \n\
  :Meta<Key>]:                 forward-paragraph() \n\
  :Meta<Key>[:                 backward-paragraph() \n\
  ~Shift Meta<Key>Delete:      delete-previous-word() \n\
   Shift Meta<Key>Delete:      backward-kill-word() \n\
  ~Shift Meta<Key>Backspace:   delete-previous-word() \n\
   Shift Meta<Key>Backspace:   backward-kill-word() \n\
  <Key>Right:                  forward-character() \n\
  <Key>Left:                   backward-character() \n\
  <Key>Down:                   next-line() \n\
  <Key>Up:                     previous-line() \n\
  <Key>Delete:                 delete-previous-character() \n\
  <Key>BackSpace:              delete-previous-character() \n\
  <Key>Linefeed:               newline-and-indent() \n\
  <Key>Return:                 newline() \n\
  <Key>:                       insert-char() \n\
  <FocusIn>:                   focus-in() \n\
  <FocusOut>:                  focus-out() \n\
  <Btn1Down>:                  select-start() \n\
  <Btn1Motion>:                extend-adjust() \n\
  <Btn1Up>:                    extend-end(PRIMARY, CUT_BUFFER0) \n\
  <Btn2Down>:                  insert-selection(PRIMARY, CUT_BUFFER0) \n\
  <Btn3Down>:                  extend-start() \n\
  <Btn3Motion>:                extend-adjust() \n\
  <Btn3Up>:                    extend-end(PRIMARY, CUT_BUFFER0) \
```

A user-supplied resource entry can use application-specific bindings, a subset of the supplied default bindings, or both. The following is an example of a user-supplied resource entry that uses a subset of the default bindings:

```
Xmh*Text.Translations: \
        <Key>Right:      forward-character() \n\
        <Key>Left:       backward-character() \n\
         Meta<Key>F:     forward-word() \n\
         Meta<Key>B:     backward-word() \n\
        :Meta<Key>]:     forward-paragraph() \n\
        :Meta<Key>[:     backward-paragraph() \n\
        <Key>:           insert-char()
```

An augmented binding that is useful with the xclipboard utility is:

```
*Text.Translations: #override \
        Button1 <Btn2Down>:      extend-end(CLIPBOARD)
```

A Text widget lets both the user and the application take control of the text being displayed. The user takes control with the scroll bar or with key strokes defined by the event bindings. The scroll bar option places the scroll bar on the left side of the widget and can be used with

any editing mode. The application takes control with procedure calls to the Text widget to:

- Display text at a specified position
- Highlight specified text areas
- Replace specified text areas

The text that is selected within a Text widget may be assigned to an X selection or copied into a cut buffer and can be retrieved by the application with the Intrinsics **XtGetSelectionValue** or the Xlib **XFetchBytes** functions respectively. Several standard selection schemes (e.g. character/word/paragraph with multi-click) are supported through the event bindings.

The class variable for the Text widget is **textWidgetClass**.

To create a Text string widget, use **XtCreateWidget** and specify the class variable **asciiStringWidgetClass**.

To create a Text file widget, use **XtCreateWidget** and specify the class variable **asciiDiskWidgetClass**.

Note

If you want to create an instance of the class **textWidgetClass**, you must provide a source and a sink when the widget is created. The Text widget cannot be instantiated without both.

When creating a Text widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	4	Border width in pixels
XtNcursor	Cursor	XC_xterm	Pointer cursor
XtNdialogHOffset	int	10	Offset of insert file dialog
XtNdialogVOffset	int	10	Offset of insert file dialog
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNdisplayPosition	int	0	Character position of first line
XtNeditType	XtEditType	XttextRead	Edit mode (see note)
XtNfile	char*	tmpnam( )	File for **asciiDiskWidgetClass**
XtNforeground	Pixel	Black	Foreground color
XtNfont	XFontStruct*	Fixed	Fontname
XtNheight	Dimension	Font height	Height of widget
XtNinsertPosition	int	0	Character position of caret
XtNleftMargin	Dimension	2	Left margin in pixels
XtNlength	int	String length	Size of the string buffer
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNselectTypes	XtTextSelectType*	See below	Selection units for multi-click
XtNsensitive	Boolean	True	Whether widget receives input
XtNstring	char*	Blank	String for **asciiStringWidgetClass**
XtNtextOptions	int	None	See below
XtNtextSink	XtTextSink	None	See below
XtNtextSource	XtTextSource	None	See below
XtNtranslations	TranslationTable	See above	event-to-action translations
XtNwidth	Dimension	100	Width of widget (pixels)
XtNx	Position	0	x coordinate in pixels
XtNy	Position	0	y coordinate in pixels

Name	Type	Default	Description

Note

1. You cannot use **XtNeditType**, **XtNfile**, **XtNlength**, and **XtNfont** with the **XtTextSetValues** and the **XtTextGetValues** calls.
2. The **XtNeditType** attribute has one of the values **XttextAppend**, **XttextEdit**, or **XttextRead**.
3. If **asciiStringWidgetClass** is used, the resource **XtNstring** specifies a buffer containing the text to be displayed and edited. **AsciiStringWidget** does not copy this buffer but uses it in-place.

The options for the **XtNtextOptions** attribute are:

Option	Description
editable	Whether or not the user is allowed to modify the text.
resizeHeight	Makes a request to the parent widget to lengthen the widget if all the text cannot fit in the window.
resizeWidth	Makes a request to the parent widget to widen the widget if the text becomes too long to fit on one line.
scrollOnOverflow	Automatically scrolls the text up when new text is entered below the bottom (last) line.
scrollVertical	Puts a scroll bar on the left side of the widget.
wordBreak	Starts a new line when a word does not fit on the current line.

These options can be ORed together to set more than one at the same time.

XtNselectionTypes is an array of entries of type **XtTextSelectType** and is used for multiclick. As the pointer button is clicked in rapid succession, each click highlights the next "type" described in the array.

XtselectAll	Selects the contents of the entire buffer.
XtselectChar	Selects text characters as the pointer moves over them.
XtselectLine	Selects the entire line.
XtselectNull	Indicates the end of the selection array.
XtselectParagraph	Selects the entire paragraph (delimited by newline characters).
XtselectPosition	Selects the current pointer position.
XtselectWord	Selects whole words (delimited by whitespace) as the pointer moves onto them.

The default selectType array is:

{XtselectPosition, XtselectWord, XtselectLine, XtselectParagraph, XtselectAll, XtselectNull}

For the default case, two rapid pointer clicks highlight the current word, three clicks highlight the current line, four clicks highlight the current paragraph, and five clicks highlight the entire text. If the timeout value is exceeded, the next pointer click returns to the first entry in the selection array. The selection array is not copied by the Text widget. The client must allocate space for the array and cannot deallocate or change it until the Text widget is destroyed or until a new selection array is set.

3.3.1. Selection Actions

The Text widget fully supports the X selection and cut buffer mechanisms. The following actions can be used to specify button bindings that will cause Text to assert ownership of one or more selections, to store the selected text into a cut buffer, and to retrieve the value of a selection or cut buffer and insert it into the text value.

insert-selection(*name*[,*name*,...])
: Retrieves the value of the first (left-most) named selection that exists or the cut buffer that is not empty and inserts it into the input stream. The specified name can be that of any selection (for example, **PRIMARY** or **SECONDARY**) or a cut buffer (i.e. **CUT_BUFFER0** through **CUT_BUFFER7**). Note that case matters.

select-start()
: Unselects any previously selected text and begins selecting new text.

select-adjust()
extend-adjust()
: Continues selecting text from the previous start position.

start-extend()
: Begins extending the selection from the farthest (left or right) edge.

select-end(*name*[,*name*,...])
extend-end(*name*[,*name*,...])
: Ends the text selection, asserts ownership of the specified selection(s) and stores the text in the specified cut buffer(s). The specified name can be that of a selection (for example, **PRIMARY** or **SECONDARY**) or a cut buffer (i.e. **CUT_BUFFER0** through **CUT_BUFFER7**). Note that case is significant. If **CUT_BUFFER0** is listed, the cut buffers are rotated before storing into buffer 0.

3.3.2. Selecting Text

To enable an application to select a piece of text, use **XtTextSetSelection**.

typedef long XtTextPosition;

void XtTextSetSelection(*w*, *left*, *right*)
    Widget *w*;
    XtTextPosition *left*, *right*;

w
: Specifies the widget ID.

left
: Specifies the character position at which the selection begins.

right
: Specifies the character position at which the selection ends.

If redisplay is not disabled, this function highlights the text and makes it the PRIMARY selection.

3.3.3. Unhighlighting Text

To unhighlight previously highlighted text in a widget, use **XtTextUnsetSelection.**

void XtTextUnsetSelection(*w*)
Widget *w*;

3.3.4. Getting Selected Text Character Positions

To enable the application to get the character positions of the selected text, use **XtTextGetSelectionPos.**

void XtTextGetSelectionPos(*w*, *pos1*, *pos2*)
Widget *w*;
XtTextPosition **pos1*, **pos2*;

w	Specifies the widget ID.
pos1	Specifies a pointer to the location to which the beginning character position of the selection is returned.
pos2	Specifies a pointer to the location to which the ending character position of the selection is returned.

If the returned values are equal, there is no current selection.

3.3.5. Replacing Text

To enable an application to replace text, use **XtTextReplace.**

int XtTextReplace(*w*, *start_pos*, *end_pos*, *text*)
Widget *w*;
XtTextPosition *start_pos*, *end_pos*;
XtTextBlock **text*;

w	Specifies the widget ID.
start_pos	Specifies the starting character position of the text replacement.
end_pos	Specifies the ending character position of the text replacement.
text	Specifies the text to be inserted into the file.

The **XtTextReplace** function deletes text in the specified range (startPos, endPos) and inserts the new text at startPos. The return value is **XawEditDone** if the replacement is successful, **XawPositionError** if the edit mode is **XttextAppend** and startPos is not the last character of the source, or **XawEditError** if either the source was read-only or the range to be deleted is larger than the length of the source.

The **XtTextBlock** structure (defined in **<X11/Text.h>** contains:

```
typedef struct {
        int firstPos;
        int length;
        char *ptr;
        Atom format;
} XtTextBlock, *TextBlockPtr;
```

The firstPos field is the starting point to use within the ptr field. The value is usually zero. The length field is the number of characters that are transferred from the ptr field. The number of characters transferred is usually the number of characters in ptr. The format field is not currently used, but should be specified as **FMT8BIT**. The **XtTextReplace** arguments

start_pos and **end_pos** represent the text source character positions for the existing text that is to be replaced by the text in the **XtTextBlock** structure. The characters from start_pos up to but not including end_pos are deleted, and the characters that are specified by the text block are inserted in their place. If start_pos and end_pos are equal, no text is deleted and the new text is inserted after start_pos.

Note

Only ASCII text is currently supported, and only one font can be used for each Text widget.

3.3.6. Redisplaying Text

To redisplay a range of characters, use **XtTextInvalidate**.

void XtTextInvalidate(*w*, *from*, *to*)
Widget *w*;
XtTextPosition *from*, *to*;

The **XtTextInvalidate** function causes the specified range of characters to be redisplayed immediately if redisplay is enabled or the next time that redisplay is enabled.

To enable redisplay, use **XtTextEnableRedisplay**.

void XtTextEnableRedisplay(*w*)
Widget *w*;

The **XtTextEnableRedisplay** function flushes any changes due to batched updates when **XtTextDisableRedisplay** was called and allows future changes to be reflected immediately.

To disable redisplay while making several changes, use **XtTextDisableRedisplay**.

void XtTextDisableRedisplay(*w*)
Widget *w*;

The **XtTextDisableRedisplay** function causes all changes to be batched until **XtTextDisplay** or **XtTextEnableRedisplay** is called.

To display batched updates, use **XtTextDisplay**.

void XtTextDisplay(*w*)
Widget *w*;

The **XtTextDisplay** function forces any accumulated updates to be displayed.

To notify the source that the length has been changed, use **XtTextSetLastPos**.

void XtTextSetLastPos(*w*, *last*);
Widget *w*;
XtTextPosition *last*;

The **XtTextSetLastPos** function notifies the text source that data has been added to or removed from the end of the source.

3.3.7. Changing Resources

The following procedures are convenience procedures that replace calls to **XtSetValues** or **XtGetValues** when only a single resource is to be modified or retrieved.

To assigns a new value to XtNtextOptions resource, use **XtTextChangeOptions**.

void XtTextChangeOptions(*w*, *options*)
    Widget *w*;
    int *options*;

To obtain the current value of XtNtextOptions for the specified widget, use **XtTextGetOptions**.

int XtTextGetOptions(*w*)
    Widget *w*;

To obtain the character position of the left-most character on the first line displayed in the widget (that is, the value of XtNdisplayPosition), use **XtTextTopPosition**.

XtTextPosition XtTextTopPosition(*w*)
    Widget *w*;

To move the insertion caret to the specified source position, use **XtTextSetInsertionPoint**.

void XtTextSetInsertionPoint(*w*, *position*)
    Widget *w*;
    XtTextPosition *position*;

The text will be scrolled vertically if necessary to make the line containing the insertion point visible. The result is equivalent to setting the XtNinsertPosition resource.

To obtain the current position of the insertion caret, use **XtTextGetInsertionPoint**.

XtTextPosition XtTextGetInsertionPoint(*w*)
    Widget *w*;

The result is equivalent to retrieving the value of the XtNinsertPosition resource.

To replace the text source in the specified widget, use **XtTextSetSource**.

void XtTextSetSource(*w*, *source*, *position*)
    Widget *w*;
    XtTextSource *source*;
    XtTextPosition *position*;

A display update will be performed if redisplay has not been disabled.

To obtain the current text source for the specified widget, use **XtTextGetSource**.

XtTextSource XtTextGetSource(*w*)
    Widget *w*;

3.3.8. Creating Sources and Sinks

The following functions for creating and destroying text sources and sinks are called automatically by **AsciiStringWidget** and **AsciiDiskWidget** and it is therefore only necessary for the client to use them when creating an instance of **textWidgetClass**.

To create a new ASCII text sink, use **XtAsciiSinkCreate**.

XtTextSink XtAsciiSinkCreate(*w*, *args*, *num_args*)
    Widget *w*;
    ArgList *args*;
    Cardinal *num_args*;

The resources required by the sink are qualified by the name and class of the parent and the sub-part name **XtNtextSink** and class **XtCTextSink**.

To deallocate an ASCII text sink, use **XtAsciiSinkDestroy**.

void XtAsciiSinkDestroy(*sink*)
    XtTextSink *sink*;

The sink must not be in use by any widget or an error will result.

To create a new text disk source, use **XtDiskSourceCreate**.

XtTextSource XtDiskSourceCreate(*w*, *args*, *num_args*)
    Widget *w*;
    ArgList *args*;
    Cardinal *num_args*;

The resources required by the source are qualified by the name and class of the parent and the sub-part name **XtNtextSource** and class **XtCTextSource**.

To deallocate a text disk source, use **XtDiskSourceDestroy**.

void XtDiskSourceDestroy(*source*)
    XtTextSource *source*;

The source must not be in use by any widget or an error will result.

To create a new text string source, use **XtStringSourceCreate**.

XtTextSource XtStringSourceCreate(*w*, *args*, *num_args*)
    Widget *w*;
    ArgList *args*;
    Cardinal *num_args*;

The resources required by the source are qualified by the name and class of the parent and the sub-part name **XtNtextSource** and class **XtCTextSource**.

To deallocate a text string source, use **XtStringSourceDestroy**.

void XtStringSourceDestroy(*source*)
    XtTextSource *source*;

The source must not be in use by any widget or an error will result.

3.4. Scrollbar Widget

The Scrollbar widget is a rectangular area that contains a slide region and a thumb (slide bar). A Scrollbar can be used alone, as a valuator, or it can be used within a composite widget (for example, a Viewport). A Scrollbar can be aligned either vertically or horizontally.

When a Scrollbar is created, it is drawn with the thumb in a contrasting color. The thumb is normally used to scroll client data and to give visual feedback on the percentage of the client data that is visible.

Each pointer button invokes a specific scroll bar action. That is, given either a vertical or horizontal alignment, the pointer button actions will scroll or return data as appropriate for that alignment. Pointer buttons 1 and 3 do not perform scrolling operations by default. Instead, they return the pixel position of the cursor on the scroll region. When pointer button 2 is clicked, the thumb moves to the current pointer position. When pointer button 2 is held down and the pointer pointer is moved, the thumb follows the pointer.

The cursor in the scroll region changes depending on the current action. When no pointer button is pressed, the cursor appears as an arrow that points in the direction that scrolling can occur. When pointer button 1 or 3 is pressed, the cursor appears as a single-headed arrow that points in the logical direction that the client will move the data. When pointer button 2 is pressed, the cursor appears as an arrow that points to the thumb.

While scrolling is in progress, the application receives notification from callback procedures. For both scrolling actions, the callback returns the Scrollbar widget ID, the client_data, and the pixel position of the pointer when the button was released. For smooth scrolling, the callback routine returns the scroll bar widget, the client data, and the current relative position of the thumb. When the thumb is moved using pointer button 2, the callback procedure is invoked continuously. When either button 1 or 3 is pressed, the callback procedure is invoked only when the button is released and the client callback procedure is responsible for moving the thumb.

The class variable for the Scrollbar widget is **scrollbarWidgetClass**.

When creating a Scrollbar widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	white	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of button border
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNforeground	Pixel	black	Thumb color
XtNheight	Dimension	See below	Height of scroll bar
XtNjumpProc	XtCallbackList	NULL	Callback for thumb select
XtNlength	Dimension	None	Major dimension (height of XtorientVertical)
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNorientation	XtOrientation	XtorientVertical	Orientation (vertical or horizontal)
XtNscrollDCursor	Cursor	XC_sb_down_arrow	Cursor for scrolling down
XtNscrollHCursor	Cursor	XC_sb_h_double_arrow	Idle horizontal cursor
XtNscrollLCursor	Cursor	XC_sb_left_arrow	Cursor for scrolling left
XtNscrollProc	XtCallbackList	NULL	Callback for the slide region
XtNscrollRCursor			

Name	Type	Default	Description
	Cursor		

XC_sb_right_arrow

Name	Type	Default	Description
			Cursor for scrolling right

Name	Type	Default	Description
XtNscrollUCursor	Cursor	XC_sb_up_arrow	Cursor for scrolling up
XtNscrollVCursor	Cursor	XC_sb_v_double_arrow	Idle vertical cursor
XtNsensitive	Boolean	True	Whether widget receives input
XtNshown	float	NULL	Percentage the thumb covers
XtNthickness	Dimension	14	Minor dimension (height if XtorientHorizontal)
XtNthumb	Pixmap	Grey	Thump pixmap
XtNtop	float	NULL	Position on scroll bar
XtNtranslations	TranslationTable	See below	Event-to-action translations
XtNwidth	Dimension	See below	Width of scroll bar
XtNx	Position	NULL	x position of scroll bar
XtNy	Position	NULL	y position of scroll bar

The class for all cursor resources is **XtCCursor**.

You can set the dimensions of the Scrollbar two ways. As for all widgets, you can use the **XtNwidth** and **XtNheight** resources. In addition, you can use an alternative method that is independent of the vertical or horizontal orientation:

XtNlength Specifies the height for a vertical Scrollbar and the width for a horizontal Scrollbar.

XtNthickness Specifies the width for a vertical Scrollbar and the height for a horizontal Scrollbar.

To create a Scrollbar widget instance, use **XtCreateWidget** and specify the class variable **scrollbarWidgetClass**.

To destroy a Scrollbar widget instance, use **XtDestroyWidget** and specify the widget ID for the Scrollbar.

The arguments to the **XtNscrollProc** callback procedure are:

void *ScrollProc*(*scrollbar*, *client_data*, *position*)
    Widget *scrollbar*;
    caddr_t *client_data*;
    caddr_t *position*; /* int */

scrollbar Specifies the ID of the Scrollbar.

client_data Specifies the client data.

position Returns the pixel position of the thumb in integer form.

The **XtNscrollProc** callback is used for incremental scrolling and is called by the **NotifyScroll** action. The position argument is a signed quantity and should be cast to an int when used. Using the default button bindings, button 1 returns a positive value, and button 3 returns a negative value. In both cases, the magnitude of the value is the distance of the pointer in pixels from the top (or left) of the Scrollbar. The value will never be less than zero or greater than the length of the Scrollbar.

The arguments to the **XtNjumpProc** callback procedure are:

void *JumpProc*(*scrollbar*, *client_data*, *percent*)
Widget *scrollbar*;
caddr_t *client_data*;
caddr_t *percent_ptr*; /* float* */

scrollbar Specifies the ID of the scroll bar widget.

client_data Specifies the client data.

percent_ptr Specifies the floating point position of the thumb (0.0 – 1.0).

The **XtNjumpProc** callback is used to implement smooth scrolling and is called by the **NotifyThumb** action. Percent_ptr must be cast to a pointer to float before use; i.e.

```
float percent = *(float*)percent_ptr;
```

With the default button bindings, button 2 moves the thumb interactively, and the **XtNjumpProc** is called on each new position of the pointer.

Note

An older interface used **XtNthumbProc** and passed the percentage by value rather than by reference. This interface is not portable across machine architectures and therefore is no longer supported but is still implemented for those (non-portable) applications which used it.

To set the position and length of a Scrollbar thumb, use **XtScrollbarSetThumb**.

void XtScrollbarSetThumb(*w*, *top*, *shown*)
Widget *w*;
float *top*;
float *shown*;

w Specifies the Scrollbar widget ID.

top Specifies the position of the top of the thumb as a fraction of the length of the Scrollbar.

shown Specifies the length of the thumb as a fraction of the total length of the Scrollbar.

XtScrollbarThumb moves the visible thumb to position (0.0 – 1.0) and length (0.0 – 1.0). Either the top or shown arguments can be specified as –1.0, in which case the current value is left unchanged. Values greater than 1.0 are truncated to 1.0.

If called from **XtNjumpProc**, **XtScrollbarSetThumb** has no effect.

The actions supported by the Scrollbar widget are:

StartScroll(*value*)
The possible values are Forward, Backward, or Continuous. This must be the first action to begin a new movement.

NotifyScroll(*value*)
The possible values are Proportional or FullLength. If the argument to StartScroll was Forward or Backward, NotifyScroll executes the **XtNscrollProc** callbacks and passes either the position of the pointer if its argument is Proportional or the full length of the scroll bar if its argument is FullLength. If the argument to StartScroll was Continuous, NotifyScroll returns without executing any callbacks.

EndScroll() This must be the last action after a movement is complete.

MoveThumb() Repositions the scroll bar thumb to the current pointer location.

NotifyThumb()
Calls the **XtNjumpProc** callbacks and passes the relative position of the pointer as a percentage of the scroll bar length.

The default bindings for Scrollbar are:

```
<Btn1Down>:     StartScroll(Forward)
<Btn2Down>:     StartScroll(Continuous) MoveThumb( ) NotifyThumb( )
<Btn3Down>:     StartScroll(Backward)
<Btn2Motion>:   MoveThumb( ) NotifyThumb( )
<BtnUp>:        NotifyScroll(Proportional) EndScroll( )
```

Examples of additional bindings a user might wish to specify in a resource file are:

```
*Scrollbar.Translations: \
    ~Meta<KeyPress>space:   StartScroll(Forward) NotifyScroll(FullLength) \n\
     Meta<KeyPress>space:   StartScroll(Backward) NotifyScroll(FullLength) \n\
                            EndScroll( )
```

3.5. Viewport Widget

The Viewport widget consists of a frame window, one or two Scrollbars, and an inner window. The frame window is determined by the viewing size of the data that is to be displayed and the dimensions to which the Viewport is created. The inner window is the full size of the data that is to be displayed and is clipped by the frame window. The Viewport widget controls the scrolling of the data directly. No application callbacks are required for scrolling.

When the geometry of the frame window is equal in size to the inner window, or when the data does not require scrolling, the Viewport widget automatically removes any scroll bars. The **forceBars** option causes the Viewport widget to display any scroll bar permanently.

The class variable for the Viewport widget is **viewportWidgetClass**.

When creating a Viewport widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNallowHoriz	Boolean	False	Flag to allow horizontal scroll bars
XtNallowVert	Boolean	False	Flag to allow vertical scroll bars
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of the border in pixels
XtNdestroyCallback	XtCallbackList	NULL	Callback for **XtDestroyWidget**
XtNforceBars	Boolean	False	Flag to force display of scroll bars
XtNheight	Dimension	height of child	Height of the widget
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNsensitive	Boolean	True	Whether widget should receive input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNuseBottom	Boolean	False	Flag to indicate bottom/top bars
XtNuseRight	Boolean	False	Flag to indicate right/left bars
XtNwidth	Dimension	width of child	Width of the widget
XtNx	Position	0	x coordinate within parent
XtNy	Position	0	y coordinate within parent

Name	Type	Default	Description

The Viewport widget manages a single child widget. When the size of the child is larger than the size of the Viewport, the user can interactively move the child within the Viewport by repositioning the Scrollbars.

The default size of the Viewport before it is realized is the width and/or height of the child. After it is realized, the viewport will allow its child to grow vertically or horizontally if **XtNallowVert** or **XtNallowHoriz** were set, respectively. If the corresponding vertical or horizontal scrolling were not enabled, the viewport will propagate the geometry request to its own parent and the child will be allowed to change size only if the (grand) parent allows it. Regardless of whether or not scrolling was enabled in the corresponding direction, if the child requests a new size smaller than the viewport size, the change will be allowed only if the parent of the viewport allows the viewport to shrink to the appropriate dimension.

To create a Viewport widget instance, use **XtCreateWidget** and specify the class variable **viewportWidgetClass**.

To insert a child into a Viewport widget, use **XtCreateWidget** and specify the widget ID of the previously created Viewport as the parent.

To remove a child from a Viewport widget, use **XtUnmanageChild** or **XtDestroyWidget** and specify the widget ID of the child.

To delete the inner window, any children, and the frame window, use **XtDestroyWidget** and specify the widget ID of the Viewport widget.

3.6. Box Widget

The Box widget provides geometry management of arbitrary widgets in a box of a specified dimension. The children are rearranged when resizing events occur either on the Box or when children are added or deleted. The Box widget always attempts to pack its children as closely as possible within the geometry allowed by its parent.

Box widgets are commonly used to manage a related set of Command widgets and are frequently called ButtonBox widgets, but the children are not limited to buttons.

The children are arranged on a background that has its own specified dimensions and color.

The class variable for the Box widget is **boxWidgetClass**.

When creating a Box widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Border width on button box
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNhSpace	Dimension	4	Pixel distance left and right of children
XtNheight	Dimension	see below	Viewing height of inner window
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNvSpace	Dimension	4	Pixel distance top and bottom of children
XtNwidth	Dimension	width of widest child	Viewing width of inner window
XtNx	Position	0	Widget location x coordinate
XtNy	Position	0	Widget location y coordinate

Name	Type	Default	Description

The Box widget positions its children in rows with **XtNhSpace** pixels to the left and right of each child and **XtNvSpace** pixels between rows. If the Box width is not specified, the Box widget uses the width of the widest child. Each time a child is managed or unmanaged, the Box widget will attempt to reposition the remaining children to compact the box. Children are positioned in order left to right, top to bottom. When the next child does not fit on the current row, a new row is started. If a child is wider than the width of the box, the box will request a larger width from it parent and will begin the layout process from the beginning if a new width is granted. After positioning all children, the Box widget attempts to shrink its own size to the minimum dimensions required for the layout.

To create a box widget instance, use **XtCreateWidget** and specify the class variable **boxWidgetClass**.

To add a child to the Box, use **XtCreateWidget** and specify the widget ID of the Box as the parent of the new widget.

To remove a child from the Box, use **XtUnmanageChild** or **XtDestroyWidget** and specify the widget ID of the child.

To destroy a Box widget instance, use **XtDestroyWidget** and specify the widget ID of the Box widget. All the children of this box are automatically destroyed at the same time.

3.7. VPaned Widget

The VPaned widget manages children in a vertically tiled fashion. A region, called a grip, appears on the border between each child. When the pointer is positioned on a grip and pressed, an arrow is displayed that indicates the significant pane that is being resized. While keeping the pointer button down, the user can move the pointer up or down. This, in turn, changes the window borders, causing one pane to shrink and some other pane to grow. The cursor indicates the pane that is of interest to the user; some other pane in the opposite direction will be chosen to grow or shrink an equal amount. The choice of alternate pane is a function of the **XtNmin**, **XtNmax** and **XtNskipAdjust** constraints on the other panes. With the default bindings, button 1 resizes the pane above the selected grip, button 3 resizes the pane below the selected grip and button 2 repositions the border between two panes only.

The class variable for the VPaned widget is **vPanedWidgetClass**.

When creating a VPaned widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNbetweenCursor	Cursor	XC_sb_left_arrow	Cursor for changing the boundary between two panes
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Border width (pixels)
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNforeground	Pixel	Black	Pixel value for the foreground color
XtNgripCursor	Cursor	XC_sb_v_double_arrow	Cursor for grip when not active
XtNgripIndent	Position	10	Offset of grip from margin (pixels)
XtNgripTranslations	TranslationTable	internal	button bindings for grip
XtNheight	Dimension	sum of child heights	Height of vPane

Name	Type	Default	Description
XtNlowerCursor	Cursor	XC_sb_down_arrow	Cursor for resizing pane below grip
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNrefigureMode	Boolean	On	Whether vPane should adjust children
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNupperCursor	Cursor	XC_sb_up_arrow	Cursor for resizing pane above grip
XtNwidth	Dimension	width of widest child	Width of vPane
XtNx	Position	0	x position of vPane
XtNy	Position	0	y position of vPane

To create a VPaned widget instance, use **XtCreateWidget** and specify the class variable **vPanedWidgetClass**.

Once the parent frame is created, you then add panes to it. Any type of widget can be paned.

To add a child pane to a VPaned frame, use **XtCreateWidget** and specify the widget ID of the VPaned widget as the parent of each new child pane.

During the creation of a child pane, the following resources, by which the VPaned widget controls the placement of the child, can be specified in the argument list or retrieved from the resource database:

Name	Type	Default	Description
XtNallowResize	Boolean	False	If False, ignore child resize requests
XtNmax	Dimension	unlimited	Maximum height for pane
XtNmin	Dimension	1	Minimum height for pane
XtNskipAdjust	Boolean	False	True if VPaned widget should not automatically resize pane

To delete a pane from a vertically paned window frame, use **XtUnmanageWidget** or **XtDestroyWidget** and specify the widget ID of the child pane.

To enable or disable a child's request for pane resizing, use **XtPanedAllowResize**.

void XtPanedAllowResize(*w*, *allow_resize*)
    Widget *w*;
    Boolean *allow_resize*;

w Specifies the widget ID of the child widget pane.

allow_resize Enables or disables a pane widget for resizing requests.

If allow_resize is **True**, VPane allows geometry requests from the child to change the pane's height. If allow_resize is **False**, VPane ignores geometry requests from the child to change the pane's height. The default state is **True** before the VPane is realized and **False** after it is realized. This procedure is equivalent to changing the **XtNallowResize** resource for the child.

To change the minimum and maximum height settings for a pane, use **XtPanedSetMinMax**.

void XtPanedSetMinMax(*w*, *min*, *max*)
    Widget *w*;
    int *min*, *max*;

w Specifies the widget ID of the child widget pane.

min New minimum height of the child, expressed in pixels.

max New maximum height of the child, expressed in pixels.

This procedure is equivalent to setting the **XtNmin** and **XtNmax** resources for the child.

To enable or disable automatic recalculation of pane sizes and positions, use **XtPanedSetRefigureMode**.

void XtPanedSetRefigureMode(*w*, *mode*)
Widget *w*;
Boolean *mode*;

w Specifies the widget ID of the VPaned widget.

mode Enables or disables refiguration.

You should set the mode to FALSE if you add multiple panes to or remove multiple panes from the parent frame after it has been realized, unless you can arrange to manage all the panes at once using **XtManageChildren**. After all the panes are added, set the mode to TRUE. This avoids unnecessary geometry calculations and "window dancing".

To delete an entire VPaned widget and all associated data structures, use **XtDestroyWidget** and specify the widget ID of the VPaned widget. All the children of the VPaned widget are automatically destroyed at the same time.

3.8. Form Widget

The Form widget can contain an arbitrary number of children or subwidgets. The Form provides geometry management for its children, which allows individual control of the position of each child. Any combination of children can be added to a Form. The initial positions of the children may be computed relative to the positions of other children. When the Form is resized, it computes new positions and sizes for its children. This computation is based upon information provided when a child is added to the Form.

The class variable for a Form widget is **formWidgetClass**.

When creating a Form widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of border in pixels
XtNdefaultDistance	int	4	Default value for XtNhorizDistance and XtNvertDistance
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNheight	Dimension	computed at realize	Height of form
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNwidth	Dimension	computed at realize	Width of form
XtNx	Position	NULL	x position of form
XtNy	Position	NULL	y position of form

To create a Form widget instance, use **XtCreateWidget** and specify the class variable **formWidgetClass**.

To add a new child to a Form, use **XtCreateWidget** and specify the widget ID of the previously created Form as the parent of the child.

When creating children that are to be added to a Form, the following additional resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbottom	XtEdgeType	XtRubber	See text
XtNfromHoriz	Widget	NULL	See text
XtNfromVert	Widget	NULL	See text
XtNhorizDistance	int	XtdefaultDistance	See text
XtNleft	XtEdgeType	XtRubber	See text
XtNresizable	Boolean	FALSE	TRUE if allowed to resize
XtNright	XtEdgeType	XtRubber	See text
XtNtop	XtEdgeType	XtRubber	See text
XtNvertDistance	int	XtdefaultDistance	See text

When a widget is added to a Form, constraints can be specified to the Form to indicate where the child should be positioned within the Form.

The resources **XtNhorizDistance** and **XtNfromHoriz** let the widget position itself a specified number of pixels horizontally away from another widget in the form. As an example, **XtNhorizDistance** could equal 10 and **XtNfromHoriz** could be the widget ID of another widget in the Form. The new widget will be placed 10 pixels to the right of the widget defined in **XtNfromHoriz**. If **XtNfromHoriz** equals NULL, then **XtNhorizDistance** is measured from the left edge of the Form.

Similarly, the resources **XtNvertDistance** and **XtNfromVert** let the widget position itself a specified number of pixels vertically away from another widget in the Form. If **XtNfromVert** equals NULL, then **XtNvertDistance** is measured from the top of the Form. Form provides a StringToWidget conversion procedure. Using this procedure, the resource database may be used to specify the **XtNfromHoriz** and **XtNfromVert** resources by widget name rather than widget id. The string value must be the name of a child of the same Form widget parent.

The **XtNtop**, **XtNbottom**, **XtNleft**, and **XtNright** resources tell the Form where to position the child when the Form is resized. **XtEdgeType** is defined in **<X11/Form.h>** and is one of **XtChainTop**, **XtChainBottom**, **XtChainLeft**, **XtChainRight** or **XtRubber**.

The values **XtChainTop**, **XtChainBottom**, **XtChainLeft**, and **XtChainRight** specify that a constant distance from an edge of the child to the top, bottom, left, and right edges respectively of the Form is to be maintained. The value **XtRubber** specifies that a proportional distance from the edge of the child to the left or top edge of the Form is to be maintained when the form is resized. The proportion is determined from the initial position of the child and the initial size of the Form. Form provides a StringToEdgeType conversion procedure to allow the resize constraints to be easily specified in a resource file.

The default width of the Form is the minimum width needed to enclose the children after computing their initial layout, with a margin of **XtNdefaultDistance** at the right and bottom edges. If a width and height is assigned to the Form that is too small for the layout, the children will be clipped by the right and bottom edges of the Form.

To remove a child from a Form, use **XtUnmanageChild** or **XtDestroyWidget** and specify the widget ID of the child widget.

To destroy a Form widget instance, use **XtDestroyWidget** and specify the widget ID of the Form. All children of the Form are automatically destroyed at the same time.

When a new child becomes managed or an old child unmanaged, Form will recalculate the positions of its children according to the values of the **XtNhorizDistance**, **XtNfromHoriz**, **XtNvertDistance** and **XtNfromVert** constraints at the time the change is made. No re-layout is performed when a child makes a geometry request.

To force or defer a re-layout of the Form, use **XtFormDoLayout**.

void XtFormDoLayout(*w*, *do_layout*)
Widget *w*;
Boolean *do_layout*;

w Specifies the Form widget.

do_layout Enables (if **True**) or disables (if **False**) layout of the Form widget.

When making several changes to the children of a Form widget after the Form has been realized, it is a good idea to disable re-layout until all changes have been made, then allow the layout. Form increments an internal count each time **XtFormDoLayout** is called with do_layout **False** and decrements the count when do_layout is **True**. When the count reaches 0, Form performs a re-layout.

3.9. Dialog Widget

The Dialog widget implements a commonly used interaction semantic to prompt for auxiliary input from a user. For example, you can use a Dialog widget when an application requires a small piece of information, such as a file name, from the user. A Dialog widget is simply a special case of the Form widget that provides a convenient way to create a "preconfigured form".

The typical Dialog widget contains three areas. The first line contains a description of the function of the Dialog widget, for example, the string "Filename:". The second line contains an area into which the user types input. The third line can contain buttons that let the user confirm or cancel the Dialog input.

The class variable for the Dialog widget is **dialogWidgetClass**.

When creating a Dialog widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of border in pixels
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNheight	Dimension	computed at create	Height of dialog
XtNlabel	String	Label name	String to be displayed
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNmaximumLength	int	256	Maximum number of input characters
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNvalue	char*	NULL	Pointer to default string
XtNwidth	Dimension	computed at create	Width of dialog
XtNx	Position	NULL	x position of dialog
XtNy	Position	NULL	y position of dialog

The instance name of the label widget within the Dialog widget is "label", and the instance name of the Dialog value widget is "value".

To create a Dialog widget instance, you can use **XtCreateWidget** and specify the class variable **dialogWidgetClass**.

To add a child button to the Dialog box, use **XtCreateWidget** and specify widget ID of the previously created Dialog box as the parent of each child. When creating buttons, you do not have to specify form constraints. The Dialog box will automatically add the constraints.

To return the character string in the text field, use **XtDialogGetValueString**.

char *XtDialogGetValueString(*w*)
Widget *w*;

w Specifies the widget ID of the Dialog box.

If a string was specified in the **XtNvalue** resource, Dialog will store the input directly into the string.

To remove a child button from the Dialog box, use **XtUnmanageChild** or **XtDestroyWidget** and specify the widget ID of the child.

To destroy a Dialog widget instance, use **XtDestroyWidget** and specify the widget ID of the Dialog widget. All children of the Dialog are automatically destroyed at the same time.

3.10. List Widget

The List widget is a rectangle that contains a list of strings formatted into rows and columns. When one of the strings is selected, it is highlighted, and an application callback routine is invoked.

The class variable for the List widget is **listWidgetClass**.

When creating a List widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of border
XtNcallback	XtCallbackList	NULL	Selection callback function
XtNcolumnSpacing	Dimension	6	Space between columns in the list
XtNcursor	Cursor	left_ptr	Pointer cursor
XtNdefaultColumns	int	2	Number of columns to use
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNfont	XFontStruct*	XtDefaultFont	Font for list text
XtNforceColumns	Boolean	False	Force the use of XtNdefaultColumns
XtNforeground	Pixel	XtDefaultForeground	Foreground (text) color
XtNheight	Dimension	Contains list exactly	Height of widget
XtNinsensitiveBorder	Pixmap	Gray	Border when not sensitive
XtNinternalHeight	Dimension	2	Spacing between list and widget edges
XtNinternalWidth	Dimension	4	Spacing between list and widget edges
XtNlist	String *	List name	An array of strings that is the list
XtNlongest	int	Longest item	Length of the longest list item in pixels
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNnumberStrings	int	Number of strings	Number of items in the list
XtNpasteBuffer	Boolean	False	Copy the selected item to cut buffer 0

Name	Type	Default	Description
XtNrowSpacing	Dimension	4	Space between rows in the list
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNverticalList	Boolean	False	Specify the layout of list items
XtNwidth	Dimension	Contains list exactly	Width of widget
XtNx	Position	0	Widget x coordinate
XtNy	Position	0	Widget y coordinate

The new resources associated with the List widget are:

XtNcolumnSpacing
XtNrowSpacing
Specify the amount of space between each of the rows and columns in the list.

XtNdefaultColumns
Specifies the default number of columns, which is used when neither the width nor the height of the List widget is specified or when XtNforceColumns is **True**.

XtNforceColumns
Specifies that the default number of columns is to be used no matter what the current size of the List widget is.

XtNheight
Specifies the height of the List widget. The default value is the minimum height that will contain the entire list with the spacing values specified. If the specified height is larger than the minimum, the list is put in the upper left corner.

XtNinternalHeight
Represents a margin, in pixels, between the top and bottom of the list and the edges of the List widget.

XtNinternalWidth
Represents a margin, in pixels, between the left and right edges of the list and the edges of the List widget.

XtNlist
Specifies the array of text strings that is to displayed in the List widget. If the default for XtNnumberStrings is used, the list must be null-terminated. If a value is not specified for the list, the number of strings is set to 1, and the name of the widget is used as the list.

XtNlongest
Specifies the length of the longest string in the current list in pixels. If the client knows the length, it should specify it. The List widget will compute a default length by searching through the list.

XtNnumberStrings
Specifies the number of strings in the current list. If a value is not specified, the list must be null-terminated.

XtNpasteBuffer
If this is **True**, then the value of the string selected will be put into X cut buffer 0.

XtNsensitive
If set to **False**, the List widget will change its window border to **XtNinsensitiveBorder** and display all items in the list as stippled strings. While the List widget is insensitive, no item in

	the list can be selected or highlighted.
XtNverticalList	If this is **True**, the elements in the list are arranged vertically; if **False**, the elements are arranged horizontally.
XtNwidth	Specifies the width of the List widget. The default value is the minimum width that will contain the entire list with the spacing values specified. If the specified width is larger than the minimum, the list is put in the upper left corner.

The List widget has three predefined actions: Set, Unset, and Notify. Set and Unset allow switching the foreground and background colors for the current list item. Notify allows processing application callbacks.

The following is the default translation table used by the List Widget:

```
<Btn1Down>,<Btn1Up>:        Set() Notify()
```

To create a List widget instance, use **XtCreateWidget** and specify the class variable **listWidgetClass**.

To destroy a List widget instance, use **XtDestroyWidget** and specify the widget ID of the List widget.

The List widget supports two callback lists:

- **XtNdestroyCallback**
- **XtNcallback**

The notify action executes the callbacks on the the **XtNcallback** list.

The call_data argument passed to callbacks on the **XtNcallback** list is a pointer to an **XtListReturnStruct** structure, defined in **<X11/List.h>**:

```
typedef struct _XtListReturnStruct {
        String string;          /* string shown in the list. */
        int index;              /* index of the item selected. */
} XtListReturnStruct;
```

3.10.1. Changing the List

To change the list that is displayed, use **XtListChange**.

void XtListChange(*w*, *list*, *nitems*, *longest*, *resize*)
    Widget *w*;
    String * *list*;
    int *nitems*, *longest*;
    Boolean *resize*;

w	Specifies the widget ID.
list	Specifies the new list for the list widget to display.
nitems	Specifies the number of items in the list. If a value less than 1 is specified, list must be null terminated.
longest	Specifies the length of the longest item in the list in pixels. If a value less than 1 is specified, the List widget calculates the value for you.
resize	Specifies a Boolean value that indicates whether the List widget should try to resize itself (**True**) or not (**False**) after making the change. Note that the

constraints of the parent of this widget are always enforced, regardless of the value specified.

XtListChange changes the list of strings that the List widget is to display.

3.10.2. Highlighting an Item

To highlight an item in the list use, **XtListHighlight**

void XtListHighlight(*w*, *item*);
Widget *w*;
int *item*;

w Specifies the widget ID.

item Specifies the index into the current list that indicates the item to be highlighted.

Only one item can be highlighted at a time. If an item is already highlighted when **XtListHighlight** is called, the highlighted item is immediately unhighlighted and the new item is highlighted.

3.10.3. Unhighlighting an Item

To unhighlight the currently highlighted item in the list, use **XtListUnhighlight**

void XtListUnhightlight(*w*);
Widget *w*;

w Specifies the widget ID.

3.10.4. Retrieving the Currently Selected Item

To retrieve an item in the list use, **XtListShowCurrent**

XtListReturnStruct *XtListShowCurrent(*w*);
Widget *w*;

w Specifies the widget ID.

The **XtListShowCurrent** function returns a pointer to an **XtListReturnStruct** structure, contains the currently highlighted item. If the value of the index member is XT_LIST_NONE, the string member is undefined, which indicates that no item is currently selected.

3.11. Grip Widget

The Grip widget provides a small region in which user input events (such as **ButtonPress**or **ButtonRelease**) may be handled. The most common use for the grip is as an attachment point for visually repositioning an object, such as the pane border in a VPaned widget.

The class variable for the Grip widget is **gripWidgetClass**.

When creating a Grip widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	0	Width of the border in pixels
XtNcallback	XtCallbackList	None	Action routine
XtNcursor	Cursor	None	Cursor for the grip
XtNdestroyCallback	XtCallbackList	NULL	Callback for **XtDestroyWidget**

Name	Type	Default	Description
XtNforeground	Pixel	XtDefaultForeground	Window background color
XtNheight	Dimension	8	Height of the widget
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNsensitive	Boolean	True	Whether widget should receive input
XtNtranslations	TranslationTable	None	Event-to-action translations
XtNwidth	Dimension	8	Width of the widget
XtNx	Position	0	x coordinate within parent
XtNy	Position	0	y coordinate within parent

Note that the Grip widget displays its region with the foreground pixel only.

The Grip widget does not declare any default event translation bindings, but it does declare a single action routine named GripAction in its action table. The client specifies an arbitrary event translation table giving parameters to the GripAction routine.

The **GripAction** action executes the callbacks on the **XtNcallback** list, passing as call_data a pointer to a **GripCallData** structure, defined in **<X11/Grip.h>**

```
typedef struct _GripCallData {
        XEvent *event;
        String *params;
        Cardinal num_params;
} GripCallDataRec, *GripCallData;
```

In this structure, the event field is a pointer to the input event that triggered the action, and params and num_params give the string parameters specified in the translation table for the particular event binding.

The following is an example of a GripAction translation table:

```
<Btn1Down>:        GripAction(press)
<Btn1Motion>:      GripAction(move)
<Btn1Up>:          GripAction(release)
```

For a complete description of the format of action routines, see the *X Toolkit Intrinsics – C Language Interface.*

To create a Grip widget instance, use **XtCreateWidget** and specify the class variable **gripWidgetClass**.

To destroy a Command button widget instance, use **XtDestroyWidget** and specify the ID of the Grip widget.

3.12. Toggle Widget

The Toggle widget is a rectanglular button that contains a text label or pixmap. This widget maintains a Boolean state (e.g. True/False or On/Off) and changes state whenever it is selected. When the pointer cursor is on the Toggle it highlights to indicate that the Toggle is available for selection. When the pointer button is pressed the Toggle is selected. This causes the state of the Toggle to reverse and its callback routine to be invoked.

Toggle buttons may also be part of a radio group. A radio group is a list of Toggle buttons in which only one Toggle may be set at any time. A radio group is identified by giving the widget id of any one of its members. There is a convenience routine, **XtToggleGetCurrent** that will return information about the Toggle in the radio group that is currently set. More information on radio groups is presented below.

The class variable for the Toggle widget is **toggleWidgetClass**.

When creating a Toggle widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	XtDefaultBackground	Window background color
XtNbackgroundPixmap	Pixmap	None	Window background pixmap
XtNbitmap	Pixmap	None	Pixmap to display in place of the label
XtNborderColor	Pixel	XtDefaultForeground	Window border color
XtNborderPixmap	Pixmap	None	Window border pixmap
XtNborderWidth	Dimension	1	Width of button border
XtNcallback	XtCallbackList	NULL	Callback for button select
XtNcursor	Cursor	None	Pointer cursor
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for **XtDestroyWidget**
XtNfont	XFontStruct*	XtDefaultFont	Label font
XtNforeground	Pixel	XtDefaultForeground	Foreground color
XtNheight	Dimension	Text height	Button height
XtNhighlightThickness	Dimension	2	Width of border to be highlighted
XtNinsensitiveBorder	Pixmap	Gray	Border when not sensitive
XtNinternalHeight	Dimension	2	Internal border height for highlighting
XtNinternalWidth	Dimension	4	Internal border width for highlighting
XtNjustify	XtJustify	XtJustifyCenter	Type of text alignment
XtNlabel	String	Button name	Button label
XtNmappedWhenManaged	Boolean	True	Whether **XtMapWidget** is automatic
XtNradioData	Pointer	Name of widget	Value that will be returned by **XtToggleGetCurrent**
XtNradioGroup	Widget	NULL	Any other widget in the Toggle's radio group
XtNresize	Boolean	True	Whether to auto-resize in **SetValues**
XtNsensitive	Boolean	True	Whether widget receives input
XtNstate	Boolean	Off	State of the Toggle widget
XtNtranslations	TranslationTable	see below	Event-to-action translations
XtNwidth	Dimension	Text width	Button width
XtNx	Position	0	x coordinate
XtNy	Position	0	y coordinate

XtNbitmap Specifies a bitmap to display in place of the text label [See the description of this resource in the Label widget for further details].

XtNcallback Specifies the callback list of functions to be called when the Toggle widget changes state. This usually occurs when the Toggle widget's notify action is called, but when a toggle is in a radio group it may change state at other times. The places where this can occur include: **XtToggleSetCurrent**, **XtToggleUnsetCurrent**, **XtToggleChangeRadioGroup**, the **set** action, **XtSetValues**, and **XtCreateWidget.**

XtNheight Specifies the height of the Toggle widget. The default value is the minimum height that will contain:
XtNinternalheight + height of **XtNlabel** + **XtNinternalHeight**
If the specified height is larger than the minimum, the label string is centered vertically.

XtNinternalHeight Represents the distance in pixels between the top and bottom of the label text or bitmap and the horizontal edges of the Toggle widget. HighlightThickness can be larger or smaller than this value.

XtNinternalWidth Represents the distance in pixels between the ends of the label text or bitmap and the vertical edges of the Toggle widget. HighlightThickness can be larger or smaller than this value.

XtNjustify Specifies left, center, or right alignment of the label string within the Toggle widget. If it is specified within an **ArgList**, one of the values **XtJustifyLeft**, **XtJustifyCenter**, or **XtJustifyRight** can be specified. In a resource of type "string", one of the values "left", "center", or "right" can be specified.

XtNlabel Specifies the text string that is to be displayed in the Toggle widget if no bitmap is specified. The default is the widget name of the Toggle widget.

XtNradioData Specifies the data that will be returned from a call to **XtToggleGetCurrent** if this widget is the one that is set in a radio group. This data is also used to identify the toggle that will be set by a call to **XtToggleSetCurrent.** The value NULL is returned by **XtToggleGetCurrent** if no widget is set in a radio group. Programmers not specify NULL as **XtNradioData**, if they intend to use **XtToggleGetCurrent**

XtNradioGroup Specifies another Toggle widget which is in the radio group to which this Toggle widget should be added. A radio group is a group of Toggle widgets, only one of which may be "set" at a time. If this value is NULL (the default) then the Toggle will not be part of any radio group and can change state without effecting any other Toggle widgets. If the widget specified in this resource is not already in a radio group then a new radio group will be created containing these two Toggle widgets. No Toggle widget can be in multiple radio groups.

XtNresize Specifies whether the Toggle widget should attempt to resize to its preferred dimensions whenever **XtSetValues** is called for it. The default is **True**.

XtNsensitive If set to **False**, the Toggle widget will change its window border to **XtNinsensitiveBorder** and will stipple the label string.

XtNstate Specifies whether the Toggle widget is set (True/On) or unset (False/Off).

XtNwidth Specifies the width of the Toggle widget. The default value is the minimum width that will contain:
XtNinternalwidth + width of **XtNlabel** + **XtNinternalWidth**
If the width is larger or smaller than the minimum, **XtNjustify**

determines how the label string is aligned.

The Toggle widget supports the following actions:

- Switching the button between the foreground and background colors with **set**, **unset** and **toggle**
- Processing application callbacks with **notify**.
- Switching the internal border between highlighted and unhighlighted states with **highlight** and **unhighlight**

The following are the default translation bindings that are used by the Toggle widget:

```
<EnterWindow>:          highlight(Always)
<LeaveWindow>:          unhighlight( )
<Btn1Down>,<Btn1Up>:    toggle( ) notify( )
```

With these bindings, the user can cancel the action before releasing the button by moving the pointer out of the Toggle widget.

3.12.1. Toggle Actions

The full list of actions supported by the Toggle widget is:

highlight(*value*) Displays the internal highlight border in the color (**XtNforeground** or **XtNbackground**) that contrasts with the interior color of the Toggle widget. This action procedure takes one of the following conditions: WhenUnset and Always. If no argument is passed then WhenUnset is assumed, this maintains backwards compatibility.

unhighlight( ) Displays the internal highlight border in the color (**XtNforeground** or **XtNbackground**) that matches the interior color of the Toggle widget.

set( ) Enters the "set" state, in which **notify** is possible and displays the interior of the button in the **XtNforeground** color. The label is displayed in the **XtNbackground** color. If the widget to be set is in a radio group then this procedure may unset another widget, which will cause all routines on its callback list to be invoked. Since only one toggle in a radio group may be set at a time the callback routines for the toggle that is to be unset will be called before the one that is to be set.

unset( ) Cancels the "set" state and displays the interior of the button in the **XtNbackground** color. The label is displayed in the **XtNforeground** color.

toggle( ) Changes the current state of the Toggle widget, causing to be set if it was previously unset, and unset if it was previously set. If the widget is to be set, and is in a radio group then this procedure may unset another widget, which will cause all routines on its callback list to be invoked. Since only one toggle in a radio group may be set at a time the callback routines for the toggle that is to be unset will be called before the one that is to be set.

reset( ) Cancels any **set** or **highlight** and displays the interior of the button in the **XtNbackground** color, with the label displayed in the **XtNforeground** color.

notify( ) Executes the **XtNcallback** callback list. The call_data contains a Boolean which is the current state of the widget.

To create a Toggle widget instance, use **XtCreateWidget** and specify the class variable **toggleWidgetClass.**

To destroy a Toggle widget instance, use **XtDestroyWidget** and specify the widget ID of the Toggle widget.

The Toggle widget supports two callbacks: **XtNdestroyCallback** and **XtNcallback.** The notify action executes the callbacks on the the **XtNcallback** list.

Changing the Toggle's Radio Group.

To enable an application to change the Toggle's current radio group, add the Toggle to a radio group, or remove the Toggle from a radio group, use **XtToggleChangeRadioGroup.**

void XtToggleChangeRadioGroup(*w*, *radio_group*)
Widget *w*, *radio_group*;

w Specifies the widget ID of the Toggle widget.

radio_group This should be any Toggle on the new radio group. If NULL then the Toggle will be removed from any radio group of which it is a member.

If a toggle is already in the set state in the new radio group, and the toggle to be added is also set then the previously set toggle in the new radio group is unset and its callback procedures are invoked.

Finding the Currently selected Toggle in a radio group of Toggles

To find the currently selected Toggle in a radio group of Toggle widgets use **XtToggleGetCurrent.**

caddr_t XtToggleGetCurrent(*radio_group*);
Widget *radio_group*;

radio_group Specifies the widget ID of any Toggle in the radio group.

The value returned by this function is the data pointed to by **XtNradioData,** for the Toggle in the radio group that is currently set. The default value for **XtNradioData** is the name of that Toggle widget. If no Toggle is set in the radio group specified then NULL is returned.

Changing the Toggle that is set in a radio group.

To change the Toggle that is currently set in a radio group use **XtToggleSetCurrent.**

void XtToggleSetCurrent(*radio_group*, *radio_data*);
Widget *radio_group*;
caddr_t *radio_data*;

radio_group Specifies the widget ID of any Toggle in the radio group.

radio_data Specifies the **XtNradioData** identifying the Toggle that should be set in the radio group specified by the *radio_group* argument.

XtToggleSetCurrent locates the Toggle widget to be set by matching *radio_data* against the **XtNradioData** for each Toggle in the radio group. If none match **XtToggleSetCurrent** returns without making any changes. If more than one Toggle matches, **XtToggleSetCurrent** will choose a Toggle to set arbitrarily. If this causes any Toggle widgets to change state all routines in their callback lists will be invoked. Since only one toggle in a radio group may be set at a time the callback routines for a Toggle that is to be unset will be called before the one that is to be set.

Unsetting all Toggles in a radio group.

To unset all Toggle widgets in a radio group use **XtToggleUnsetCurrent.**

void XtToggleUnsetCurrent(*radio_group*);
Widget *radio_group*;

radio_group Specifies the widget ID of any Toggle in the radio group.

If this causes a Toggle widget to change state all routines on its callback list will be invoked.

3.13. Template Widget - Creating A Custom Widget

Although the task of creating a new widget may at first appear a little daunting, there is a basic simple pattern that all widgets follow. The Athena widget library contains three files that are intended to assist in writing a custom widget.

Reasons for wishing to write a custom widget include:

- Convenient access to resource management procedures to obtain fonts, colors, etc., even if user customization is not desired.
- Convenient access to user input dispatch and translation management procedures.
- Access to callback mechanism for building higher-level application libraries.
- Customizing the interface or behavior of an existing widget to suit a special application need.
- Desire to allow user customization of resources such as fonts, colors, etc., or to allow convenient re-binding of keys and buttons to internal functions.
- Converting a non-Toolkit application to use the Toolkit.

In each of these cases, the operation needed to create a new widget is to "subclass" an existing one. If the desired semantics of the new widget are similar to an existing one, then the implementation of the existing widget should be examined to see how much work would be required to create a subclass that will then be able to share the existing class methods. Much time will be saved in writing the new widget if an existing widget class Expose, Resize and/or GeometryManager method can be shared by the subclass.

Note that some trivial uses of a ''bare-bones'' widget may be achieved by simply creating an instance of the Core widget. The class variable to use when creating a Core widget is **widgetClass**. The geometry of the Core widget is determined entirely by the parent widget.

It is very often the case than an application will have a special need for a certain set of functions and that many copies of these functions will be needed. For example, when converting an older application to use the Toolkit, it may be desireable to have a "Window Widget" class that might have the following semantics:

- Allocate 2 drawing colors in addition to a background color.
- Allocate a text font.
- Execute an application-supplied function to handle exposure events.
- Execute an application-supplied function to handle user input events.

It is obvious that a completely general-purpose WindowWidgetClass could be constructed that would export all class methods as callbacks lists, but such a widget would be very large and would have to choose some arbitrary number of resources such as colors to allocate. An application that used many instances of the general-purpose widget would therefore un-necessarily waste many resources.

In this section, an outline will be given of the procedure to follow to construct a special-purpose widget to address the items listed above. The reader should refer to the appropriate sections of the *X Toolkit Intrinsics – C Language Interface* for complete details of the material outlined here. Section 1.4 of the *Intrinsics* should be read in conjunction with this section.

All Athena widgets have three separate files associated with them:

- A "public" header file containing declarations needed by applications programmers
- A "private" header file containing additional declarations needed by the widget and any subclasses
- A source code file containing the implementation of the widget

This separation of functions into three files is suggested for all widgets, but nothing in the Toolkit actually requires this format. In particular, a private widget created for a single application may easily combine the "public" and "private" header files into a single file, or merge the contents into another application header file. Similarly, the widget implementation can be merged into other application code.

In the following example, the public header file **<X11/Template.h>**, the private header file **<X11/TemplateP.h>** and the source code file **<X11/Template.c>** will be modified to produce the "WindowWidget" described above. In each case, the files have been designed so that a global string replacement of ''Template'' and ''template'' with the name of your new widget, using the appropriate case, can be done.

3.13.1. Public Header File

The public header file contains declarations that will be required by any application module that needs to refer to the widget; whether to create an instance of the class, to perform an **XtSetValues** operation, or to call a public routine implemented by the widget class.

The contents of the Template public header file, **<X11/Template.h>**, are:

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

/* XConsortium: Template.h,v 1.2 88/10/25 17:22:09 swick Exp $ */
/* Copyright Massachusetts Institute of Technology 1987, 1988 */

#ifndef _Template_h
#define _Template_h

/***********************************************************************
 *
 * Template widget
 *
 ***********************************************************************/

/* Resources:

 Name                 Class                RepType     Default Value
 ----                 -----                -------     -------------
 background           Background           Pixel       XtDefaultBackground
 border               BorderColor          Pixel       XtDefaultForeground
 borderWidth          BorderWidth          Dimension   1
 destroyCallback      Callback             Pointer     NULL
 height               Height               Dimension   0
 mappedWhenManaged    MappedWhenManaged    Boolean     True
 sensitive            Sensitive            Boolean     True
 width                Width                Dimension   0
 x                    Position             Position    0
```

```
    y                   Position            Position        0

    */

    /* define any special resource names here that are not in <X11/StringDefs.h> */

    #define XtNtemplateResource             "templateResource"

    #define XtCTemplateResource             "TemplateResource"

    /* declare specific TemplateWidget class and instance datatypes */

    typedef struct _TemplateClassRec*       TemplateWidgetClass;
    typedef struct _TemplateRec*            TemplateWidget;

    /* declare the class constant */

    extern WidgetClass templateWidgetClass;

    #endif _Template_h
```

You will notice that most of this file is documentation. The crucial parts are the last 8 lines where macros for any private resource names and classes are defined and where the widget class datatypes and class record pointer are declared.

For the "WindowWidget", we want 2 drawing colors, a callback list for user input and an **XtNexposeCallback** callback list, and we will declare three convenience procedures, so we need to add

```
/* Resources:
    ...
callback            Callback        Callback        NULL
drawingColor1       Color           Pixel           XtDefaultForeground
drawingColor2       Color           Pixel           XtDefaultForeground
exposeCallback      Callback        Callback        NULL
font                Font            XFontStruct*    XtDefaultFont
    ...
*/

#define XtNdrawingColor1                "drawingColor1"
#define XtNdrawingColor2                "drawingColor2"
#define XtNexposeCallback               "exposeCallback"

extern Pixel WindowColor1(/* Widget */);
extern Pixel WindowColor2(/* Widget */);
extern Font  WindowFont( /* Widget */);
```

Note that we have chosen to call the input callback list by the generic name, **XtNcallback**, rather than a specific name. If widgets that define a single user-input action all choose the same resource name then there is greater possibility for an application to switch between widgets of different types.

3.13.2. Private Header File

The private header file contains the complete declaration of the class and instance structures for the widget and any additional private data that will be required by anticipated subclasses of the widget. Information in the private header file is normally hidden from the application and is designed to be accessed only through other public procedures; e.g. **XtSetValues**.

The contents of the Template private header file, **<X11/TemplateP.h>**, are:

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

/* XConsortium: TemplateP.h,v 1.2 88/10/25 17:31:47 swick Exp $ */
/* Copyright Massachusetts Institute of Technology 1987, 1988 */

#ifndef _TemplateP_h
#define _TemplateP_h

#include "Template.h"
/* include superclass private header file */
#include <X11/CoreP.h>

/* define unique representation types not found in <X11/StringDefs.h> */

#define XtRTemplateResource                "TemplateResource"

typedef struct {
    int empty;
} TemplateClassPart;

typedef struct _TemplateClassRec {
    CoreClassPart       core_class;
    TemplateClassPart   template_class;
} TemplateClassRec;

extern TemplateClassRec templateClassRec;

typedef struct {
    /* resources */
    char* resource;
    /* private state */
} TemplatePart;

typedef struct _TemplateRec {
    CorePart            core;
    TemplatePart        template;
} TemplateRec;

#endif  _TemplateP_h
```

The private header file includes the private header file of its superclass, thereby exposing the entire internal structure of the widget. It may not always be advantageous to do this; your own project development style will dictate the appropriate level of detail to expose in each module.

The "WindowWidget" needs to declare two fields in its instance structure to hold the drawing colors, a resource field for the font and a field for the expose and user input callback lists:

```
typedef struct {
    /* resources */
    Pixel color_1;
    Pixel color_2;
    XFontStruct* font;
    XtCallbackList expose_callback;
    XtCallbackList input_callback;
    /* private state */
    /* (none) */
} WindowPart;
```

3.13.3. Widget Source File

The source code file implements the widget class itself. The unique part of this file is the declaration and initialization of the widget class record structure and the declaration of all resources and action routines added by the widget class.

The contents of the Template implementation file, **<X11/Template.c>**, are:

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

/* XConsortium: Template.c,v 1.2 88/10/25 17:40:25 swick Exp $ */
/* Copyright Massachusetts Institute of Technology 1987, 1988 */

#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include "TemplateP.h"

static XtResource resources[] = {
#define offset(field) XtOffset(TemplateWidget, template.field)
    /* {name, class, type, size, offset, default_type, default_addr}, */
    { XtNtemplateResource, XtCTemplateResource, XtRTemplateResource, sizeof(char*),
                        offset(resource), XtRString, "default" },
#undef offset
};

static void TemplateAction(/* Widget, XEvent*, String*, Cardinal* */);

static XtActionsRec actions[] =
{
    /* {name,           procedure}, */
    {"template",        TemplateAction},
};

static char translations[] =
"    <Key>:             template( ) \n\
";

TemplateClassRec templateClassRec = {
  { /* core fields */
    /* superclass       */      (WidgetClass) &widgetClassRec,
    /* class_name       */      "Template",
    /* widget_size      */      sizeof(TemplateRec),
    /* class_initialize */      NULL,
    /* class_part_initialize*/  NULL,
    /* class_inited     */      FALSE,
    /* initialize       */      NULL,
    /* initialize_hook  */      NULL,
    /* realize          */      XtInheritRealize,
    /* actions          */      actions,
    /* num_actions      */      XtNumber(actions),
    /* resources        */      resources,
    /* num_resources    */      XtNumber(resources),
    /* xrm_class        */      NULLQUARK,
    /* compress_motion  */      TRUE,
    /* compress_exposure*/      TRUE,
    /* compress_enterleave*/    TRUE,
    /* visible_interest */      FALSE,
    /* destroy          */      NULL,
    /* resize           */      NULL,
    /* expose           */      NULL,
    /* set_values       */      NULL,
    /* set_values_hook  */      NULL,
    /* set_values_almost*/      XtInheritSetValuesAlmost,
    /* get_values_hook  */      NULL,
    /* accept_focus     */      NULL,
```

```
  /* version              */          XtVersion,
     /* callback_private   */          NULL,
     /* tm_table           */          translations,
     /* query_geometry     */          XtInheritQueryGeometry,
     /* display_accelerator */         XtInheritDisplayAccelerator,
     /* extension          */          NULL
  },
  { /* template fields */
     /* empty              */          0
  }
};

WidgetClass templateWidgetClass = (WidgetClass)&templateClassRec;
```

The resource list for the "WindowWidget" might look like the following:

```
static XtResource resources[] = {
#define offset(field) XtOffset(WindowWidget, window.field)
     /* {name, class, type, size, offset, default_type, default_addr}, */
     { XtNdrawingColor1, XtCColor, XtRPixel, sizeof(Pixel),
                         offset(color_1), XtRString, XtDefaultForeground },
     { XtNdrawingColor2, XtCColor, XtRPixel, sizeof(Pixel),
                         offset(color_2), XtRString, XtDefaultForeground },
     { XtNfont, XtCFont, XtRFontStruct, sizeof(XFontStruct*),
                         offset(font), XtRString, XtDefaultFont },
     { XtNexposeCallback, XtCCallback, XtRCallback, sizeof(XtCallbackList),
                         offset(expose_callback), XtRCallback, NULL },
     { XtNcallback, XtCCallback, XtRCallback, sizeof(XtCallbackList),
                         offset(input_callback), XtRCallback, NULL },
#undef offset
};
```

The user input callback will be implemented by an action procedure which passes the event pointer as call_data. The action procedure is declared as:

```
/* ARGSUSED */
static void InputAction(w, event, params, num_params)
     Widget w;
     XEvent *event;
     String *params;                    /* unused */
     Cardinal *num_params;              /* unused */
{
     XtCallCallbacks(w, XtNcallback, (caddr_t)event);
}

static XtActionsRec actions[] =
{
     /* {name,          procedure}, */
     {"input",          InputAction},
};
```

and the default input binding will be to execute the input callbacks on **KeyPress** and **ButtonPress**:

```
static char translations[] =
"    <Key>:             input( ) \n\
     <BtnDown>:         input( ) \
";
```

In the class record declaration and initialization, the only field that is different from the Template is the expose procedure:

```
/* ARGSUSED */
static void Redisplay(w, event, region)
    Widget w;
    XEvent *event;      /* unused */
    Region region;
{
    XtCallCallbacks(w, XtNexposeCallback, (caddr_t)region);
}

WindowClassRec windowClassRec = {

    ...

    /* expose           */          Redisplay,
```

The "WindowWidget" will also declare three public procedures to return the drawing colors and the font id, saving the application the effort of constructing an argument list for a call to **XtGetValues**:

```
Pixel WindowColor1(w)
    Widget w;
{
    return ((WindowWidget)w)->window.color_1;
}

Pixel WindowColor2(w)
    Widget w;
{
    return ((WindowWidget)w)->window.color_2;
}

Font WindowFont(w)
    Widget w;
{
    return ((WindowWidget)w)->window.font->fid;
}
```

The "WindowWidget" is now complete. The application can retrieve the two drawing colors from the widget instance by calling either **XtGetValues**, or the **WindowColor** functions. The actual window created for the "WindowWidget" is available by calling the **XtWindow** function.

To test the new ''WindowWidget'', you may substitute ''window'' for ''command'' in the sample program given in Section 2.7.3.

Index

INTRINSICS *MAN* PAGES

CHAPTER TWO

NAME

XwarrowWidgetClass – the X Widget's arrow drawing widget

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Arrow.h>

CLASSES

The Arrow widget is built from the Core and XwPrimitive classes.

The widget class to use when creating an arrow is **XwarrowWidgetClass.** The class name for this widget is **Arrow.**

DESCRIPTION

The Arrow widget supports drawing of an arrow within the bounds of its window. It uses the primitive widget's border highlighting routines.

The arrow can be drawn in the directions of up, down, left and right. The Arrow widget also supports two types of callbacks: Button selections, and Button releases.

NEW RESOURCES

The Arrow widget defines a set of resources used by the programmer to specify the data for the arrow. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by the Arrow widget.

Arrow Resource Set			
Name	**Class**	**Type**	**Default**
XtNarrowDirection	XtCArrowDirection	int	up

XtNarrowDirection

This resource is the means by which the arrow direction is set. It can be defined in either of two ways: Through the .Xdefaults file by the strings "up", "down", "left" and "right". Within an arg list for use in XtSetValues() by the defines XwARROW_UP, XwARROW_DOWN, XwARROW_LEFT and XwARROW_RIGHT.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

Input to the Arrow widget is driven by the mouse buttons. The Primitive class resources of XtNselect and XtNrelease define the callback lists used by the Arrow widget. Thus, to receive input from an arrow, the application adds callbacks to the arrow using these two resource types. The default translation set for the Arrow widget is as follows.

<Btn1Down>:	select()	
<Btn1Up>:	release()	
<EnterWindow>:	enter()	
<LeaveWindow>:	leave()	
<KeyDown>Select:	select()	HP "Select" key
<KeyUp>Select:	unselect()	HP "Select" key

ACTIONS

select:

Selections occurring on an arrow cause the arrow to be displayed as selected and its primitive XtNselect callbacks are called.

release:

Release redraws the arrow in its normal mode and calls its primitive XtNrelease callbacks.

enter:

If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be highlighted. Otherwise no action is taken.

leave:

If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWCREATETILE(3X)

NAME

XwbulletinWidgetClass – the X Widgets bulletin board manager widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/BBoard.h>
```

CLASSES

The bulletin board manager widget is built from the Core, Composite, Constraint and XwManager classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no resources that the user can set, there is no table for Composite class resources.

The widget class to use when creating a bulletin board is **XwbulletinWidgetClass.** The class name is **BulletinBoard.**

DESCRIPTION

The bulletin board manager widget is a composite widget that enforces no ordering on its children. It is up to the application to specify the x and y coordinates of the children inserted into this widget, otherwise they will all appear at (0,0).

This manager widget supports 3 different layout policies: minimize (the default), maximize and ignore. When the layout policy is set to minimize, the manager will create a box that is just large enough to contain all of its children, regardless of any provided width and height values. The ignore setting forces the manager to honor its given width and height, it will not grow or shrink in response to the addition, deletion or altering of its children. When set to the maximize setting, the BulletinBoard widget will ask for additional space when it needs it, but will not give up extra space.

The bulletin board manager also implements the X Widgets keyboard interface.

No callbacks are defined for this manager.

NEW RESOURCES

The bulletin board manager widget class does not define any additional resources; all necessary resources are present in its superclasses. The programmer should refer to the man pages for the bulletin board's superclasses to determine the resources that can be set and the defaults settings for these resources.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

SEE ALSO

XtCallCallbacks(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAddGrab, XtRemoveGrab – redirect user input to a modal widget

SYNTAX

void XtAddGrab(*w*, *exclusive*, *spring_loaded*)
Widget *w*;
Boolean *exclusive*;
Boolean *spring_loaded*;

void XtRemoveGrab(*w*)
Widget *w*;

ARGUMENTS

exclusive	Specifies whether user events should be dispatched exclusively to this widget or also to previous widgets in the cascade.
spring_loaded	Specifies whether this widget was popped up because the user pressed a pointer button.
w	Specifies the widget to add to or remove from the modal cascade.

DESCRIPTION

The **XtAddGrab** function appends the widget (and associated parameters) to the modal cascade and checks that exclusive is **True** if spring_loaded is **True**. If these are not **True**, **XtAddGrab** generates an error.

The modal cascade is used by **XtDispatchEvent** when it tries to dispatch a user event. When at least one modal widget is in the widget cascade, **XtDispatchEvent** first determines if the event should be delivered. It starts at the most recent cascade entry and follows the cascade up to and including the most recent cascade entry added with the exclusive parameter **True**.

This subset of the modal cascade along with all descendants of these widgets comprise the active subset. User events that occur outside the widgets in this subset are ignored or remapped. Modal menus with submenus generally add a submenu widget to the cascade with exclusive **False**. Modal dialog boxes that need to restrict user input to the most deeply nested dialog box add a subdialog widget to the cascade with exclusive **True**. User events that occur within the active subset are delivered to the appropriate widget, which is usually a child or further descendant of the modal widget.

Regardless of where on the screen they occur, remap events are always delivered to the most recent widget in the active subset of the cascade that has spring_loaded **True**, if any such widget exists.

The **XtRemoveGrab** function removes widgets from the modal cascade starting at the most recent widget up to and including the specified widget. It issues an error if the specified widget is not on the modal cascade.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppAddActions – register an action table

SYNTAX

void XtAppAddActions(*app_context, actions, num_actions*)
XtAppContext *app_context;*
XtActionList *actions;*
Cardinal *num_actions;*

ARGUMENTS

app_context	Specifies the application context.
actions	Specifies the action table to register.
num_args	Specifies the number of entries in this action table.

DESCRIPTION

The **XtAppAddActions** function adds the specified action table and registers it with the translation manager.

SEE ALSO

XtParseTranslationTable(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

XtAppAddConverter (3Xt)

NAME

XtAppAddConverter – register resource converter

SYNTAX

void XtAppAddConverter(*app_context, from_type, to_type, converter, convert_args, num_args*)
XtAppContext *app_context;*
String *from_type;*
String *to_type;*
XtConverter *converter;*
XtConvertArgList *convert_args;*
Cardinal *num_args;*

ARGUMENTS

app_context	Specifies the application context.
converter	Specifies the type converter procedure.
convert_args	Specifies how to compute the additional arguments to the converter or NULL.
from_type	Specifies the source type.
num_args	Specifies the number of additional arguments to the converter or zero.
to_type	Specifies the destination type.

DESCRIPTION

The **XtAppAddConverter** registers a the specified resource converter.

SEE ALSO

XtConvert(3Xt), XtStringConversionWarning(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppAddInput, XtRemoveInput – register and remove an input source

SYNTAX

XtInputId XtAppAddInput(*app_context, source, condition, proc, client_data*)
XtAppContext *app_context;*
int *source;*
caddr_t *condition;*
XtInputCallbackProc *proc;*
caddr_t *client_data;*

void XtRemoveInput(*id*)
XtInputId *id;*

ARGUMENTS

app_context	Specifies the application context that identifies the application.
client_data	Specifies the argument that is to be passed to the specified procedure when input is available.
condition	Specifies the mask that indicates a read, write, or exception condition or some operating system dependent condition.
id	Specifies the ID returned from the corresponding **XtAppAddInput** call.
proc	Specifies the procedure that is to be called when input is available.
source	Specifies the source file descriptor on a UNIX-based system or other operating system dependent device specification.

DESCRIPTION

The **XtAppAddInput** function registers with the Intrinsics read routine a new source of events, which is usually file input but can also be file output. Note that file should be loosely interpreted to mean any sink or source of data. **XtAppAddInput** also specifies the conditions under which the source can generate events. When input is pending on this source, the callback procedure is called.

The legal values for the condition argument are operating-system dependent. On a UNIX-based system, the condition is some union of **XtInputReadMask**, **XtInputWriteMask**, and **XtInputExceptMask**. The **XtRemoveInput** function causes the Intrinsics read routine to stop watching for input from the input source.

SEE ALSO

XtAppAddTimeOut(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppAddTimeOut, XtRemoveTimeOut – register and remove timeouts

SYNTAX

XtIntervalId XtAppAddTimeOut(*app_context, interval, proc, client_data*)
XtAppContext *app_context;*
unsigned long *interval;*
XtTimerCallbackProc *proc;*
caddr_t *client_data;*

void XtRemoveTimeOut(*timer*)
XtIntervalId *timer;*

ARGUMENTS

app_context	Specifies the application context for which the timer is to be set.
client_data	Specifies the argument that is to be passed to the specified procedure when input is available.
interval	Specifies the time interval in milliseconds.
proc	Specifies the procedure that is to be called when time expires.
timer	Specifies the ID for the timeout request to be destroyed.

DESCRIPTION

The **XtAppAddTimeOut** function creates a timeout and returns an identifier for it. The timeout value is set to interval. The callback procedure is called when the time interval elapses, and then the timeout is removed.

The **XtRemoveTimeOut** function removes the timeout. Note that timeouts are automatically removed once they trigger.

SEE ALSO

XtAppAddInput(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppAddWorkProc, XtRemoveWorkProc – Add and remove background processing procedures

SYNTAX

XtWorkProcId XtAppAddWorkProc(*app_context, proc, client_data*)
XtAppContext *app_context;*
XtWorkProc *proc;*
caddr_t *client_data;*

void XtRemoveWorkProc(*id*)
XtWorkProcId *id;*

ARGUMENTS

app_context	Specifies the application context that identifies the application.
client_data	Specifies the argument that is to be passed to the specified procedure when it is called.
proc	Specifies the procedure that is to be called when time expires.
id	Specifies which work procedure to remove.

DESCRIPTION

The **XtAppAddWorkProc** function adds the specified work procedure for the application identified by app_context.

The **XtRemoveWorkProc** function explicitly removes the specified background work procedure.

SEE ALSO

XtAppNextEvent(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppCreateShell – create top-level widget instance

SYNTAX

Widget XtAppCreateShell(*application_name, application_class, widget_class, display, args, num_args*)
String *application_name;*
String *application_class;*
WidgetClass *widget_class;*
Display **display;*
ArgList *args;*
Cardinal *num_args;*

ARGUMENTS

application_class Specifies the class name of this application.

application_name
Specifies the name of the application instance.

args Specifies the argument list in which to set in the WM_COMMAND property.

display Specifies the display from which to get the resources.

num_args Specifies the number of arguments in the argument list.

widget_class Specifies the widget class that the application top-level widget should be.

DESCRIPTION

The **XtAppCreateShell** function saves the specified application name and application class for qualifying all widget resource specifiers. The application name and application class are used as the left-most components in all widget resource names for this application. **XtAppCreateShell** should be used to create a new logical application within a program or to create a shell on another display. In the first case, it allows the specification of a new root in the resource hierarchy. In the second case, it uses the resource database associated with the other display.

Note that the widget returned by **XtAppCreateShell** has the WM_COMMAND property set for session managers (see Chapter 4).

SEE ALSO

XtCreateWidget(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppError, XtAppSetErrorHandler, XtAppSetWarningHandler, XtAppWarning – low-level error handlers

SYNTAX

void XtAppError(*app_context, message*)
XtAppContext *app_context;*
String *message;*

void XtAppSetErrorHandler(*app_context, handler*)
XtAppContext *app_context;*
XtErrorHandler *handler;*

void XtAppSetWarningHandler(*app_context, handler*)
XtAppContext *app_context;*
XtErrorHandler *handler;*

void XtAppWarning(*app_context, message*)
XtAppContext *app_context;*
String *message;*

ARGUMENTS

app_context	Specifies the application context.
message	Specifies the nonfatal error message that is to be reported.
handler	Specifies the new fatal error procedure, which should not return, or the nonfatal error procedure, which usually returns.
message	Specifies the message that is to be reported.

DESCRIPTION

The **XtAppError** function calls the installed error procedure and passes the specified message.

The **XtAppSetErrorHandler** function registers the specified procedure, which is called when a fatal error condition occurs.

The **XtAppSetWarningHandler** registers the specified procedure, which is called when a nonfatal error condition occurs.

The **XtAppWarning** function calls the installed nonfatal error procedure and passes the specified message.

SEE ALSO

XtAppGetErrorDatabase(3Xt), XtAppErrorMsg(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppErrorMsg, XtAppSetErrorMsgHandler, XtAppSetWarningMsgHandler, XtAppWarningMsg – high-level error handlers

SYNTAX

void XtAppErrorMsg(*app_context, name, type, class, default, params, num_params*)
XtAppContext *app_context;*
String *name;*
String *type;*
String *class;*
String *default;*
String **params;*
Cardinal **num_params;*

void XtAppSetErrorMsgHandler(*app_context, msg_handler*)
XtAppContext *app_context;*
XtErrorMsgHandler *msg_handler;*

void XtAppSetWarningMsgHandler(*app_context, msg_handler*)
XtAppContext *app_context;*
XtErrorMsgHandler *msg_handler;*

void XtAppWarningMsg(*app_context, name, type, class, default, params, num_params*)
XtAppContext *app_context;*
String *name;*
String *type;*
String *class;*
String *default;*
String **params;*
Cardinal **num_params;*

ARGUMENTS

app_context	Specifies the application context.
class	Specifies the resource class.
default	Specifies the default message to use.
name	Specifies the general kind of error.
type	Specifies the detailed name of the error.
msg_handler	Specifies the new fatal error procedure, which should not return or the nonfatal error procedure, which usually returns.
num_params	Specifies the number of values in the parameter list.
params	Specifies a pointer to a list of values to be stored in the message.

DESCRIPTION

The **XtAppErrorMsg** function calls the high-level error handler and passes the specified information.

The **XtAppSetErrorMsgHandler** function registers the specified procedure, which is called when a fatal error occurs.

The **XtAppSetWarningMsgHandler** function registers the specified procedure, which is called when a nonfatal error condition occurs.

The **XtAppWarningMsg** function calls the high-level error handler and passes the specified information.

SEE ALSO

XtAppGetErrorDatabase(3Xt), XtAppError(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppGetErrorDatabase, XtAppGetErrorDatabaseText – obtain error database

SYNTAX

XrmDatabase *XtAppGetErrorDatabase(*app_context*)
XtAppContext *app_context;*

void XtAppGetErrorDatabaseText(*app_context, name, type, class, default, buffer_return, nbytes, database*)
XtAppContext *app_context;*
char **name, *type, *class;*
char **default;*
char **buffer_return;*
int *nbytes;*
XrmDatabase *database;*

ARGUMENTS

app_context	Specifies the application context.
buffer_return	Specifies the buffer into which the error message is to be returned.
class	Specifies the resource class of the error message.
database	Specifies the name of the alternative database that is to be used or NULL if the application's database is to be used.
default	Specifies the default message to use.
name	
type	Specifies the name and type that are concatenated to form the resource name of the error message.
nbytes	Specifies the size of the buffer in bytes.

DESCRIPTION

The **XtAppGetErrorDatabase** function returns the address of the error database. The Intrinsics do a lazy binding of the error database and do not merge in the database file until the first call to **XtAppGetErrorDatbaseText**.

The **XtAppGetErrorDatabaseText** returns the appropriate message from the error database or returns the specified default message if one is not found in the error database.

SEE ALSO

XtAppError(3Xt), XtAppErrorMsg(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppGetSelectionTimeout, XtAppSetSelectionTimeout – set and obtain selection timeout values

SYNTAX

unsigned long XtAppGetSelectionTimeout(*app_context*)
XtAppContext *app_context*;

void XtAppSetSelectionTimeout(*app_context, timeout*)
XtAppContext *app_context*;
unsigned long *timeout*;

ARGUMENTS

app_context	Specifies the application context.
timeout	Specifies the selection timeout in milliseconds.

DESCRIPTION

The **XtAppGetSelectionTimeout** function returns the current selection timeout value, in milliseconds. The selection timeout is the time within which the two communicating applications must respond to one another. The initial timeout value is set by the **selectionTimeout** application resource, or, if **selectionTimeout** is not specified, it defaults to five seconds.

The **XtAppSetSelectionTimeout** function sets the Intrinsics's selection timeout mechanism. Note that most applications should not set the selection timeout.

SEE ALSO

XtOwnSelection(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAppNextEvent, XtAppPending, XtAppPeekEvent, XtAppProcessEvent, XtDispatchEvent, XtAppMainLoop – query and process events and input

SYNTAX

void XtAppNextEvent(*app_context, event_return*)
XtAppContext *app_context;*
XEvent **event_return;*

Boolean XtAppPeekEvent(*app_context, event_return*)
XtAppContext *app_context;*
XEvent **event_return;*

XtInputMask XtAppPending(*app_context*)
XtAppContext *app_context;*

void XtAppProcessEvent(*app_context, mask*)
XtAppContext *app_context;*
XtInputMask *mask;*

Boolean XtDispatchEvent(*event*)
XEvent **event;*

void XtAppMainLoop(*app_context*)
XtAppContext *app_context;*

ARGUMENTS

app_context	Specifies the application context that identifies the application .
event	Specifies a pointer to the event structure that is to be dispatched to the appropriate event handler.
event_return	Returns the event information to the specified event structure.
mask	Specifies what types of events to process. The mask is the bitwise inclusive OR of any combination of **XtIMXEvent, XtIMTimer,** and **XtIMAlternateInput**. As a convenience, the X Toolkit defines the symbolic name **XtIMAll** to be the bitwise inclusive OR of all event types.

DESCRIPTION

If no input is on the X input queue, **XtAppNextEvent** flushes the X output buffer and waits for an event while looking at the other input sources and timeout values and calling any callback procedures triggered by them. This wait time can be used for background processing (see Section 7.8).

If there is an event in the queue, **XtAppPeekEvent** fills in the event and returns a nonzero value. If no X input is on the queue, **XtAppPeekEvent** flushes the output buffer and blocks until input is available (possibly calling some timeout callbacks in the process). If the input is an event, **XtAppPeekEvent** fills in the event and returns a nonzero value. Otherwise, the input is for an alternate input source, and **XtAppPeekEvent** returns zero.

The **XtAppPending** function returns a nonzero value if there are events pending from the X server, timer pending, or other input sources pending. The value returned is a bit mask that is the OR of **XtIMXEvent, XtIMTimer,** and **XtIMAlternateInput** (see **XtAppProcessEvent**). If there are no events pending, **XtAppPending** flushes the output buffer and returns zero.

The **XtAppProcessEvent** function processes one timer, alternate input, or X event. If there is nothing of the appropriate type to process, **XtAppProcessEvent** blocks until there is. If there is more than one type of thing available to process, it is undefined

XtAppNextEvent (3Xt)

which will get processed. Usually, this procedure is not called by client applications (see **XtAppMainLoop**). **XtAppProcessEvent** processes timer events by calling any appropriate timer callbacks, alternate input by calling any appropriate alternate input callbacks, and X events by calling **XtDispatchEvent**.

When an X event is received, it is passed to **XtDispatchEvent**, which calls the appropriate event handlers and passes them the widget, the event, and client-specific data registered with each procedure. If there are no handlers for that event registered, the event is ignored and the dispatcher simply returns. The order in which the handlers are called is undefined.

The **XtDispatchEvent** function sends those events to the event handler functions that have been previously registered with the dispatch routine. **XtDispatchEvent** returns **True** if it dispatched the event to some handler and **False** if it found no handler to dispatch the event to. The most common use of **XtDispatchEvent** is to dispatch events acquired with the **XtAppNextEvent** procedure. However, it also can be used to dispatch user-constructed events. **XtDispatchEvent** also is responsible for implementing the grab semantics for **XtAddGrab**.

The **XtAppMainLoop** function first reads the next incoming X event by calling **XtAppNextEvent** and then it dispatches the event to the appropriate registered procedure by calling **XtDispatchEvent**. This constitutes the main loop of X Toolkit applications, and, as such, it does not return. Applications are expected to exit in response to some user action. There is nothing special about **XtAppMainLoop**; it is simply an infinite loop that calls **XtAppNextEvent** and then **XtDispatchEvent**.

Applications can provide their own version of this loop, which tests some global termination flag or tests that the number of top-level widgets is larger than zero before circling back to the call to **XtAppNextEvent**.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtBuildEventMask – retrieve a widget's event mask

SYNTAX

EventMask XtBuildEventMask(*w*)
Widget *w*;

ARGUMENTS

w Specifies the widget.

DESCRIPTION

The **XtBuildEventMask** function returns the event mask representing the logical OR of all event masks for event handlers registered on the widget with **XtAddEventHandler** and all event translations, including accelerators, installed on the widget. This is the same event mask stored into the **XSetWindowAttributes** structure by **XtRealizeWidget** and sent to the server when event handlers and translations are installed or removed on the realized widget.

SEE ALSO

XtAddEventHandler(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtCallAcceptFocus – call a widget's accept_focus procedure

SYNTAX

Boolean XtCallAcceptFocus(*w*, *time*)
    Widget *w*;
    Time **time*;

ARGUMENTS

time	Specifies the X time of the event that is causing the accept focus.
w	Specifies the widget.

DESCRIPTION

The **XtCallAcceptFocus** function calls the specified widget's accept_focus procedure, passing it the specified widget and time, and returns what the accept_focus procedure returns. If accept_focus is NULL, **XtCallAcceptFocus** returns **False**.

SEE ALSO

XtSetKeyboardFocus(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtCallCallbacks, XtHasCallbacks – process callbacks

SYNTAX

void XtCallCallbacks(*w, callback_name, call_data*)
Widget *w;*
String *callback_name;*
caddr_t *call_data;*

typedef enum {XtCallbackNoList, XtCallbackHasNone, XtCallbackHasSome} XtCallbackStatus;

XtCallbackStatus XtHasCallbacks(*w, callback_name*)
Widget *w;*
String *callback_name;*

ARGUMENTS

callback_name	Specifies the callback list to be executed or checked.
call_data	Specifies a callback-list specific data value to pass to each of the callback procedure in the list.
w	Specifies the widget.

DESCRIPTION

The **XtCallCallbacks** function calls each procedure that is registered in the specified widget's callback list.

The **XtHasCallbacks** function first checks to see if the widget has a callback list identified by callback_name. If the callback list does not exist, **XtHasCallbacks** returns **XtCallbackNoList**. If the callback list exists but is empty, it returns **XtCallbackHasNone**. If the callback list exists and has at least one callback registered, it returns **XtCallbackHasSome**.

SEE ALSO

XtAddCallback(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtClass, XtSuperClass, XtIsSubclass, XtCheckSubclass, XtIsComposite, XtIsManaged – obtain and verify a widget's class

SYNTAX

WidgetClass XtClass(*w*)
Widget *w*;

WidgetClass XtSuperclass(*w*)
Widget *w*;

Boolean XtIsSubclass(*w*, *widget_class*)
Widget *w*;
WidgetClass *widget_class*;

void XtCheckSubclass(*w*, *widget_class*, *message*)
Widget *w*;
WidgetClass *widget_class*;
String *message*;

Boolean XtIsComposite(*w*)
Widget *w*;

Boolean XtIsManaged(*w*)
Widget *w*;

ARGUMENTS

w	Specifies the widget.
widget_class	Specifies the widget class that the application top-level widget should be.
message	Specifies the message that is to be used.

DESCRIPTION

The **XtClass** function returns a pointer to the widget's class structure.

The **XtSuperclass** function returns a pointer to the widget's superclass class structure.

The **XtIsSubclass** function returns **True** if the class of the specified widget is equal to or is a subclass of the specified widget class. The specified widget can be any number of subclasses down the chain and need not be an immediate subclass of the specified widget class. Composite widgets that need to restrict the class of the items they contain can use **XtIsSubclass** to find out if a widget belongs to the desired class of objects.

The **XtCheckSubclass** macro determines if the class of the specified widget is equal to or is a subclass of the specified widget class. The widget can be any number of subclasses down the chain and need not be an immediate subclass of the specified widget class. If the specified widget is not a subclass, **XtCheckSubclass** constructs an error message from the supplied message, the widget's actual class, and the expected class and calls **XtErrorMsg**. **XtCheckSubclass** should be used at the entry point of exported routines to ensure that the client has passed in a valid widget class for the exported operation.

XtCheckSubclass is only executed when the widget has been compiled with the compiler symbol DEBUG defined; otherwise, it is defined as the empty string and generates no code.

The **XtIsComposite** function is a convenience function that is equivalent to **XtIsSubclass** with **compositeWidgetClass** specified.

The **XtIsManaged** macro (for widget programmers) or function (for application programmers) returns **True** if the specified child widget is managed or **False** if it is not.

SEE ALSO

XtAppErrorMsg(3Xt), XtDisplay(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtConfigureWidget, XtMoveWidget, XtResizeWidget – move and resize widgets

SYNTAX

void XtConfigureWidget(*w, x, y, width, height, border_width*)
Widget *w;*
Position *x;*
Position *y;*
Dimension *width;*
Dimension *height;*
Dimension *border_width;*

void XtMoveWidget(*w, x, y*)
Widget *w;*
Position *x;*
Position *y;*

void XtResizeWidget(*w, width, height, border_width*)
Widget *w;*
Dimension *width;*
Dimension *height;*
Dimension *border_width;*

void XtResizeWindow(*w*)
Widget *w;*

ARGUMENTS

width *height* *border_width*	Specify the new widget size.
w	Specifies the widget.
x *y*	Specify the new widget x and y coordinates.

DESCRIPTION

The **XtConfigureWidget** function returns immediately if the specified geometry fields are the same as the old values. Otherwise, **XtConfigureWidget** writes the new x, y, width, height, and border_width values into the widget and, if the widget is realized, makes an Xlib **XConfigureWindow** call on the widget's window.

If either the new width or height is different from its old value, **XtConfigureWidget** calls the widget's resize procedure to notify it of the size change; otherwise, it simply returns.

The **XtMoveWidget** function returns immediately if the specified geometry fields are the same as the old values. Otherwise, **XtMoveWidget** writes the new x and y values into the widget and, if the widget is realized, issues an Xlib **XMoveWindow** call on the widget's window.

The **XtResizeWidget** function returns immediately if the specified geometry fields are the same as the old values. Otherwise, **XtResizeWidget** writes the new width, height, and border_width values into the widget and, if the widget is realized, issues an **XConfigureWindow** call on the widget's window.

If the new width or height are different from the old values, **XtResizeWidget** calls the widget's resize procedure to notify it of the size change.

The **XtResizeWindow** function calls the **XConfigureWindow** Xlib function to make the window of the specified widget match its width, height, and border width. This request is done unconditionally because there is no way to tell if these values match

the current values. Note that the widget's resize procedure is not called.

There are very few times to use **XtResizeWindow**; instead, you should use **XtResizeWidget**.

SEE ALSO

XtMakeGeometryRequest(3Xt), XtQueryGeometry(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtConvert, XtDirectConvert – invoke resource converters

SYNTAX

void XtConvert(*w, from_type, from, to_type, to_return*)
Widget *w;*
String *from_type;*
XrmValuePtr *from;*
String *to_type;*
XrmValuePtr *to_return;*

void XtDirectConvert(*converter, args, num_args, from, to_return*)
XtConverter *converter;*
XrmValuePtr *args;*
Cardinal *num_args;*
XrmValuePtr *from;*
XrmValuePtr *to_return;*

ARGUMENTS

args	Specifies the argument list that contains the additional arguments needed to perform the conversion (often NULL).
converter	Specifies the conversion procedure that is to be called.
from	Specifies the value to be converted.
from_type	Specifies the source type.
num_args	Specifies the number of additional arguments (often zero).
to_type	Specifies the destination type.
to_return	Returns the converted value.
w	Specifies the widget to use for additional arguments (if any are needed).

DESCRIPTION

The **XtConvert** function looks up the type converter registered to convert from_type to to_type, computes any additional arguments needed, and then calls **XtDirectConvert**.

The **XtDirectConvert** function looks in the converter cache to see if this conversion procedure has been called with the specified arguments. If so, it returns a descriptor for information stored in the cache; otherwise, it calls the converter and enters the result in the cache.

Before calling the specified converter, **XtDirectConvert** sets the return value size to zero and the return value address to NULL. To determine if the conversion was successful, the client should check to_return.address for non-NULL.

SEE ALSO

XtAppAddConverter(3Xt), XtStringConversionWarning(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtCreateApplicationContext, XtDestroyApplicationContext, XtWidgetToApplicationContext, XtToolkitInitialize – create, destroy, and obtain an application context

SYNTAX

XtAppContext XtCreateApplicationContext()

void XtDestroyApplicationContext(*app_context*)
XtAppContext *app_context*;

XtAppContext XtWidgetToApplicationContext(*w*)
Widget *w*;

void XtToolkitInitialize()

ARGUMENTS

app_context	Specifies the application context.
w	Specifies the widget to use for additional arguments (if any are needed).

DESCRIPTION

The **XtCreateApplicationContext** function returns an application context, which is an opaque type. Every application must have at least one application context.

The **XtDestroyApplicationContext** function destroys the specified application context as soon as it is safe to do so. If called from with an event dispatch (for example, a callback procedure), **XtDestroyApplicationContext** does not destroy the application context until the dispatch is complete.

The **XtWidgetToApplicationContext** function returns the application context for the specified widget.

The semantics of calling **XtToolkitInitialize** more than once are undefined.

SEE ALSO

XtDisplayInitialize(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtCreatePopupShell – creates a popup shell

SYNTAX

Widget XtCreatePopupShell(*name, widget_class, parent, args, num_args*)
String *name;*
WidgetClass *widget_class;*
Widget *parent;*
ArgList *args;*
Cardinal *num_args;*

ARGUMENTS

args	Specifies the argument list to override the resource defaults.
name	Specifies the text name for the created shell widget.
num_args	Specifies the number of arguments in the argument list.
parent	Specifies the parent widget.
widget_class	Specifies the widget class pointer for the created shell widget.

DESCRIPTION

The **XtCreatePopupShell** function ensures that the specified class is a subclass of **Shell** and, rather than using insert_child to attach the widget to the parent's children list, attaches the shell to the parent's pop-ups list directly.

A spring-loaded pop-up invoked from a translation table already must exist at the time that the translation is invoked, so the translation manager can find the shell by name. Pop-ups invoked in other ways can be created "on-the-fly" when the pop-up actually is needed. This delayed creation of the shell is particularly useful when you pop up an unspecified number of pop-ups. You can look to see if an appropriate unused shell (that is, not currently popped up) exists and create a new shell if needed.

SEE ALSO

XtCreateWidget(3Xt), XtPopdown(3Xt), XtPopup(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtCreateWidget, XtCreateManagedWidget, XtDestroyWidget – create and destroy widgets

SYNTAX

Widget XtCreateWidget(*name, widget_class, parent, args, num_args*)
String *name;*
WidgetClass *widget_class;*
Widget *parent;*
ArgList *args;*
Cardinal *num_args;*

Widget XtCreateManagedWidget(*name, widget_class, parent, args, num_args*)
String *name;*
WidgetClass *widget_class;*
Widget *parent;*
ArgList *args;*
Cardinal *num_args;*

void XtDestroyWidget(*w*)
Widget *w;*

ARGUMENTS

args	Specifies the argument list to override the resource defaults.
name	Specifies the resource name for the created widget, which is used for retrieving resources and, for that reason, should not be the same as any other widget that is a child of same parent.
num_args	Specifies the number of arguments in the argument list.
parent	Specifies the parent widget.
w	Specifies the widget.
widget_class	Specifies the widget class pointer for the created widget.

DESCRIPTION

The **XtCreateWidget** function performs much of the boilerplate operations of widget creation:

- Checks to see if the class_initialize procedure has been called for this class and for all superclasses and, if not, calls those necessary in a superclass-to-subclass order.
- Allocates memory for the widget instance.
- If the parent is a subclass of **constraintWidgetClass**, it allocates memory for the parent's constraints and stores the address of this memory into the constraints field.
- Initializes the core nonresource data fields (for example, parent and visible).
- Initializes the resource fields (for example, background_pixel) by using the resource lists specified for this class and all superclasses.
- If the parent is a subclass of **constraintWidgetClass**, it initializes the resource fields of the constraints record by using the constraint resource list specified for the parent's class and all superclasses up to **constraintWidgetClass.**
- Calls the initialize procedures for the widget by starting at the **Core** initialize procedure on down to the widget's initialize procedure.

- If the parent is a subclass of **compositeWidgetClass**, it puts the widget into its parent's children list by calling its parent's insert_child procedure. For further information, see Section 3.5.
- If the parent is a subclass of **constraintWidgetClass**, it calls the constraint initialize procedures, starting at **constraintWidgetClass** on down to the parent's constraint initialize procedure.

Note that you can determine the number of arguments in an argument list by using the **XtNumber** macro. For further information, see Section 11.1.

The **XtCreateManagedWidget** function is a convenience routine that calls **XtCreateWidget** and **XtManageChild**.

The **XtDestroyWidget** function provides the only method of destroying a widget, including widgets that need to destroy themselves. It can be called at any time, including from an application callback routine of the widget being destroyed. This requires a two-phase destroy process in order to avoid dangling references to destroyed widgets.

In phase one, **XtDestroyWidget** performs the following:

- If the being_destroyed field of the widget is **True**, it returns immediately.
- Recursively descends the widget tree and sets the being_destroyed field to **True** for the widget and all children.
- Adds the widget to a list of widgets (the destroy list) that should be destroyed when it is safe to do so.

Entries on the destroy list satisfy the invariant that if w2 occurs after w1 on the destroy list then w2 is not a descendent of w1. (A descendant refers to both normal and pop-up children.)

Phase two occurs when all procedures that should execute as a result of the current event have been called (including all procedures registered with the event and translation managers), that is, when the current invocation of **XtDispatchEvent** is about to return or immediately if not in **XtDispatchEvent**.

In phase two, **XtDestroyWidget** performs the following on each entry in the destroy list:

- Calls the destroy callback procedures registered on the widget (and all descendants) in post-order (it calls children callbacks before parent callbacks).
- If the widget's parent is a subclass of **compositeWidgetClass** and if the parent is not being destroyed, it calls **XtUnmanageChild** on the widget and then calls the widget's parent's delete_child procedure (see Section 3.4).
- If the widget's parent is a subclass of **constraintWidgetClass**, it calls the constraint destroy procedure for the parent, then the parent's superclass, until finally it calls the constraint destroy procedure for **constraintWidgetClass**.
- Calls the destroy methods for the widget (and all descendants) in post-order. For each such widget, it calls the destroy procedure declared in the widget class, then the destroy procedure declared in its superclass, until finally it calls the destroy procedure declared in the Core class record.
- Calls **XDestroyWindow** if the widget is realized (that is, has an X window). The server recursively destroys all descendant windows.
- Recursively descends the tree and deallocates all pop-up widgets, constraint records, callback lists and, if the widget is a subclass of **compositeWidgetClass**, children.

SEE ALSO

XtAppCreateShell(3Xt), XtCreatePopupShell(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtCreateWindow – window creation convenience function

SYNTAX

void XtCreateWindow(*w, window_class, visual, value_mask, attributes*)
Widget *w;*
unsigned int *window_class;*
Visual **visual;*
XtValueMask *value_mask;*
XSetWindowAttributes **attributes;*

ARGUMENTS

attributes	Specifies the window attributes to use in the **XCreateWindow** call.
value_mask	Specifies which attribute fields to use.
visual	Specifies the visual type (usually **CopyFromParent**).
w	Specifies the widget that is used to set the x,y coordinates and so on.
window_class	Specifies the Xlib window class (for example, **InputOutput, InputOnly**, or **CopyFromParent**).

DESCRIPTION

The **XtCreateWindow** function calls the Xlib **XCreateWindow** function with values from the widget structure and the passed parameters. Then, it assigns the created window to the widget's window field.

XtCreateWindow evaluates the following fields of the **Core** widget structure:

- depth
- screen
- parent -> core.window
- x
- y
- width
- height
- border_width

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtDisplay, XtParent, XtScreen, XtWindow – obtain window information about a widget

SYNTAX

```
Display *XtDisplay(w)
    Widget w;

Widget XtParent(w)
    Widget w;

Screen *XtScreen(w)
    Widget w;

Window XtWindow(w)
    Widget w;
```

ARGUMENTS

w Specifies the widget.

DESCRIPTION

XtDisplay returns the display pointer for the specified widget.

XtParent returns the parent widget for the specified widget.

XtScreen returns the screen pointer for the specified widget.

XtWindow returns the window of the specified widget.

SEE ALSO

XtClass(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtDisplayInitialize, XtOpenDisplay, XtDatabase, XtCloseDisplay – initialize, open, or close a display

SYNTAX

void XtToolkitInitialize()

void XtDisplayInitialize(*app_context, display, application_name, application_class, options, num_options, argc, argv*)
XtAppContext *app_context;*
Display **display;*
String *application_name;*
String *application_class;*
XrmOptionDescRec **options;*
Cardinal *num_options;*
Cardinal **argc;*
String **argv;*

Display *XtOpenDisplay(*app_context, display_string, application_name, application_class, options, num_options, argc, argv*)
XtAppContext *app_context;*
String *display_string;*
String *application_name;*
String *application_class;*
XrmOptionDescRec **options;*
Cardinal *num_options;*
Cardinal **argc;*
String **argv;*

void XtCloseDisplay(*display*)
Display **display;*

XrmDatabase XtDatabase(*display*)
Display **display;*

ARGUMENTS

argc	Specifies a pointer to the number of command line parameters.
argv	Specifies the command line parameters.
app_context	Specifies the application context.
application_class	Specifies the class name of this application, which usually is the generic name for all instances of this application.
application_name	Specifies the name of the application instance.
display	Specifies the display. Note that a display can be in at most one application context.
num_options	Specifies the number of entries in the options list.
options	Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to **XrmParseCommand**. For further information, see *Xlib – C Language X Interface.*

DESCRIPTION

The **XtDisplayInitialize** function builds the resource database, calls the Xlib **XrmParseCommand** function to parse the command line, and performs other per display

initialization. After **XrmParseCommand** has been called, argc and argv contain only those parameters that were not in the standard option table or in the table specified by the options argument. If the modified argc is not zero, most applications simply print out the modified argv along with a message listing the allowable options. On UNIX-based systems, the application name is usually the final component of argv[0]. If the synchronize resource is **True** for the specified application, **XtDisplayInitialize** calls the Xlib **XSynchronize** function to put Xlib into synchronous mode for this display connection. If the reverseVideo resource is **True**, the Intrinsics exchange **XtDefaultForeground** and **XtDefaultBackground** for widgets created on this display. (See Section 9.6.1).

The **XtOpenDisplay** function calls **XOpenDisplay** the specified display name. If display_string is NULL, **XtOpenDisplay** uses the current value of the –display option specified in argv and if no display is specified in argv, uses the user's default display (on UNIX-based systems, this is the value of the DISPLAY environment variable).

If this succeeds, it then calls **XtDisplayInitialize** and pass it the opened display and the value of the –name option specified in argv as the application name. If no name option is specified, it uses the application name passed to **XtOpenDisplay**. If the application name is NULL, it uses the last component of argv[0]. **XtOpenDisplay** returns the newly opened display or NULL if it failed.

XtOpenDisplay is provided as a convenience to the application programmer.

The **XtCloseDisplay** function closes the specified display as soon as it is safe to do so. If called from within an event dispatch (for example, a callback procedure), **XtCloseDisplay** does not close the display until the dispatch is complete. Note that applications need only call **XtCloseDisplay** if they are to continue executing after closing the display; otherwise, they should call **XtDestroyApplicationContext** or just exit.

The **XtDatabase** function returns the fully merged resource database that was built by **XtDisplayInitialize** associated with the display that was passed in. If this display has not been initialized by **XtDisplayInitialize**, the results are not defined.

SEE ALSO

XtAppCreateShell(3Xt), XtCreateApplicationContext(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtGetGC, XtReleaseGC – obtain and destroy a sharable GC

SYNTAX

GC XtGetGC(*w, value_mask, values*)
Widget *w;*
XtGCMask *value_mask;*
XGCValues **values;*

void XtReleaseGC(*w, gc*)
Widget *w;*
GC *gc;*

ARGUMENTS

gc	Specifies the GC to be deallocated.
values	Specifies the actual values for this GC.
value_mask	Specifies which fields of the values are specified.
w	Specifies the widget.

DESCRIPTION

The **XtGetGC** function returns a sharable, read-only GC. The parameters to this function are the same as those for **XCreateGC** except that a widget is passed instead of a display. **XtGetGC** shares only GCs in which all values in the GC returned by **XCreateGC** are the same. In particular, it does not use the value_mask provided to determine which fields of the GC a widget considers relevant. The value_mask is used only to tell the server which fields should be filled in with widget data and which it should fill in with default values. For further information about value_mask and values, see **XCreateGC** in the *Xlib – C Language X Interface.*

The **XtReleaseGC** function deallocate the specified shared GC.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtGetSelectionValue, XtGetSelectionValues – obtain selection values

SYNTAX

void XtGetSelectionValue(*w, selection, target, callback, client_data, time*)
Widget *w;*
Atom *selection;*
Atom *target;*
XtSelectionCallbackProc *callback;*
caddr_t *client_data;*
Time *time;*

void XtGetSelectionValues(*w, selection, targets, count, callback, client_data, time*)
Widget *w;*
Atom *selection;*
Atom **targets;*
int *count;*
XtSelectionCallbackProc *callback;*
caddr_t *client_data;*
Time *time;*

ARGUMENTS

callback	Specifies the callback procedure that is to be called when the selection value has been obtained.
client_data	Specifies the argument that is to be passed to the specified procedure when it is called.
client_data	Specifies the client data (one for each target type) that is passed to the callback procedure when it is called for that target.
count	Specifies the length of the targets and client_data lists.
selection	Specifies the particular selection desired (that is, primary or secondary).
target	Specifies the type of the information that is needed about the selection.
targets	Specifies the types of information that is needed about the selection.
time	Specifies the timestamp that indicates when the selection value is desired.
w	Specifies the widget that is making the request.

DESCRIPTION

The **XtGetSelectionValue** function requests the value of the selection that has been converted to the target type. The specified callback will be called some time after **XtGetSelectionValue** is called; in fact, it may be called before or after **XtGetSelectionValue** returns.

The **XtGetSelectionValues** function is similar to **XtGetSelectionValue** except that it takes a list of target types and a list of client data and obtains the current value of the selection converted to each of the targets. The effect is as if each target were specified in a separate call to **XtGetSelectionValue**. The callback is called once with the corresponding client data for each target. **XtGetSelectionValues** does guarantee that all the conversions will use the same selection value becaues the ownership of the selection cannot change in the middle of the list, as would be when calling **XtGetSelectionValue** repeatedly.

SEE ALSO

XtAppGetSelectionTimeout(3Xt), XtOwnSelection(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtGetSubresources, XtGetApplicationResources – obtain subresources or application resources

SYNTAX

void XtGetSubresources(*w, base, name, class, resources, num_resources, args, num_args*)
Widget *w;*
caddr_t *base;*
String *name;*
String *class;*
XtResourceList *resources;*
Cardinal *num_resources;*
ArgList *args;*
Cardinal *num_args;*

void XtGetApplicationResources(*w, base, resources, num_resources, args, num_args*)
Widget *w;*
caddr_t *base;*
XtResourceList *resources;*
Cardinal *num_resources;*
ArgList *args;*
Cardinal *num_args;*

ARGUMENTS

args	Specifies the argument list to override resources obtained from the resource database.
base	Specifies the base address of the subpart data structure where the resources should be written.
class	Specifies the class of the subpart.
name	Specifies the name of the subpart.
num_args	Specifies the number of arguments in the argument list.
num_resources	Specifies the number of resources in the resource list.
resources	Specifies the resource list for the subpart.
w	Specifies the widget that wants resources for a subpart or that identifies the resource database to search.

DESCRIPTION

The **XtGetSubresources** function constructs a name/class list from the application name/class, the name/classes of all its ancestors, and the widget itself. Then, it appends to this list the name/class pair passed in. The resources are fetched from the argument list, the resource database, or the default values in the resource list. Then, they are copied into the subpart record. If args is NULL, num_args must be zero. However, if num_args is zero, the argument list is not referenced.

The **XtGetApplicationResources** function first uses the passed widget, which is usually an application shell, to construct a resource name and class list, Then, it retrieves the resources from the argument list, the resource database, or the resource list default values. After adding base to each address, **XtGetApplicationResources** copies the resources into the address given in the resource list. If args is NULL, num_args must be zero. However, if num_args is zero, the argument list is not referenced. The portable way to specify application resources is to declare them as members of a structure and pass the address of the structure as the base argument.

SEE ALSO

XtGetResourceList(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtGetResourceList – obtain resource list

SYNTAX

void XtGetResourceList(*class*, *resources_return*, *num_resources_return*);
WidgetClass *class*;
XtResourceList **resources_return*;
Cardinal **num_resources_return*;

ARGUMENTS

num_resources_return
Specifies a pointer to where to store the number of entries in the resource list.

resources_return Specifies a pointer to where to store the returned resource list. The caller must free this storage using **XtFree** when done with it.

widget_class Specifies the widget class.

DESCRIPTION

If it is called before the widget class is initialized (that is, before the first widget of that class has been created), **XtGetResourceList** returns the resource list as specified in the widget class record. If it is called after the widget class has been initialized, **XtGetResourceList** returns a merged resource list that contains the resources for all superclasses.

SEE ALSO

XtGetSubresources(3Xt), XtOffset(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtMalloc, XtCalloc, XtRealloc, XtFree, XtNew, XtNewString – memory management functions

SYNTAX

char *XtMalloc(*size*);
    Cardinal *size*;

char *XtCalloc(*num*, *size*);
    Cardinal *num*;
    Cardinal *size*;

char *XtRealloc(*ptr*, *num*);
    char **ptr*;
    Cardinal *num*;

void XtFree(*ptr*);
    char **ptr*;

type *XtNew(*type*);
    type;

String XtNewString(*string*);
    String *string*;

ARGUMENTS

num	Specifies the number of bytes or array elements.
ptr	Specifies a pointer to the old storage or to the block of storage that is to be freed.
size	Specifies the size of an array element (in bytes) or the number of bytes desired.
string	Specifies a previously declared string.
type	Specifies a previously declared data type.

DESCRIPTION

The **XtMalloc** functions returns a pointer to a block of storage of at least the specified size bytes. If there is insufficient memory to allocate the new block, **XtMalloc** calls **XtErrorMsg**.

The **XtCalloc** function allocates space for the specified number of array elements of the specified size and initializes the space to zero. If there is insufficient memory to allocate the new block, **XtCalloc** calls **XtErrorMsg**.

The **XtRealloc** function changes the size of a block of storage (possibly moving it). Then, it copies the old contents (or as much as will fit) into the new block and frees the old block. If there is insufficient memory to allocate the new block, **XtRealloc** calls **XtErrorMsg**. If ptr is NULL, **XtRealloc** allocates the new storage without copying the old contents; that is, it simply calls **XtMalloc**.

The **XtFree** function returns storage and allows it to be reused. If ptr is NULL, **XtFree** returns immediately.

XtNew returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, **XtNew** calls **XtErrorMsg**. **XtNew** is a convenience macro that calls **XtMalloc** with the following arguments specified:

```
((type *) XtMalloc((unsigned) sizeof(type))
```

XtNewString returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, **XtNewString** calls **XtErrorMsg**. **XtNewString** is a convenience macro that calls **XtMalloc** with the following arguments specified:

(strcpy(XtMalloc((unsigned) strlen(str) + 1), str))

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XwmenupaneWidgetClass – the X Widgets menupane meta widget.

SYNOPSIS

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>

CLASSES

The menupane widget class is built from the Core, Composite, Constraint and XwManager classes.

DESCRIPTION

The menupane class is an X Widget meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other menupane widget classes. It provides a collection of resources which will be needed by most menupane subclasses.

NEW RESOURCES

The MenuPane defines a set of resource types used by the programmer to specify the data for widgets which are subclasses of MenuPane. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name.

MenuPane Resource Set			
Name	**Class**	**Type**	**Default**
XtNtitleShowing	XtCTitleShowing	Boolean	TRUE
XtNmgrTitleOverride	XtCTitleOverride	Boolean	FALSE
XtNtitleType	XtCTitleType	int	XwSTRING
XtNtitleString	XtCTitleString	String	widget name
XtNtitleImage	XtCTitleImage	XImage *	NULL
XtNfont	XtCFont	XFontStruct *	"fixed"
XtNattachTo	XtCAttachTo	String	NULL
XtNmnemonic	XtCMnemonic	String	NULL
XtNselect	XtCCallback	Pointer	NULL

XtNtitleShowing
This resource may be used by the application to control the displaying of a title within the menupane. This may be overridden, however, by a menu manager using the XtNmgrTitleOverride resource.

XtNmgrTitleOverride
This resource is not intended to be used by applications; it should only be used by a menu manager widget, for overriding the application, and forcing off the menupane title. This is useful for those menu managers whose style dictates that certain menupane should not have a title displayed.

XtNtitleType
Two styles of titles are supported by the MenuPane widget. They include text string titles and image titles. To programmatically set this resource, use either the XwSTRING define or the XwIMAGE define. To set this resource using the .Xdefaults file, use one of the strings string or image.

XtNtitleString
If the title type resource indicates that a title string should be displayed, then this resource will contain the title string which is to be used. In the case where the application does not specify a title string, the name of the menupane widget will be used. The title is displayed using the foreground color.

XtNtitleImage
If the title type resource indicates that a title image should be displayed, then this resource will contain a pointer to an XImage structure; this structure describes the title image data.

XtNfont
If the title type resource indicates that a title string should be displayed, then this resource will describe the font used to draw the title string.

XtNattachTo
When used in conjunction with a menu manager, this resource provides the means by which the menupane may be attached as a cascade to a menubutton. The string which is specified represents the name of the menubutton to which the menupane is to be attached; this provides the means by which the menu manager is able to construct the menu tree. To specify that this menupane should be treated as the top level menupane within the menu tree, this string should contain the name of the menu manager widget, instead of a menubutton widget. Specifying a NULL string indicates that the menupane will not be presently attached to anything. If the menupane does not have a menu manager associated with it, then this resource is unused.

XtNmnemonic
Certain menu managers allow some of their menupanes to have a mnemonic. Mnemonics may be used to post a menupane using the keyboard, instead of using the pointer device. This resource is a NULL terminated string, containing a single character. Typically, the character is the same as one present in the menupane title.

XtNselect
This resource provides the means for registering callback routines which will be invoked when the menupane receives a select action.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X)

NAME

XwmenuSepWidgetClass – the X Widget's menu item separator widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/MenuSep.h>
```

CLASSES

MenuSep is built from the Core, XwPrimitive, XwButton, and XwMenuBtn classes.

The widget class to use when creating a menu separator widget is **XwmenuSepWidgetClass.**

The class name for this widget is **MenuSep.**

DESCRIPTION

The MenuSep widget is a primitive widget to be used as an item separator placed between items in a menu. Several different line drawing styles are provided.

NEW RESOURCES

The MenuSep widget defines a one additional resource type. The programmer can also set the values for the Core and Primitive resources to set attributes for this widget. The Button and MenuButton resources are unused for this widget.

MenuSep Resource Set			
Name	**Class**	**Type**	**Default**
XtNseparatorType	XtCseparatorType	int	XwSINGLE_LINE

XtNseparatorType

This resource defines the type of line drawing to be done in the menu separator widget. Five different line drawing styles are provided. They are single, double, single dashed, double dashed and no line. The separator type can be set through an argument list by using one of the defines: XwSINGLE_LINE, XwDOUBLE_LINE, XwSINGLE_DASHED_LINE, XwDOUBLE_DASHED_LINE, and XwNO_LINE. The separator type can be set through the .Xdefaults file by using one of the following strings: single_line, double_line single_dashed_line, double_dashed_line and no_line.

The line drawing done within the menu separator will be automatically centered within the height of the widget.

The separator type of no_line is provided as an escape to the application programmer who needs a different style of drawing. To create an alternate style, a pixmap the height of the widget can be created. After the separator widget has been created, this pixmap can be used as the background pixmap by building an argument list using the XtNbackgroundPixmap argument type as defined by Core and setting the widgets background through XtSetValues. Whenever the widget is redrawn its background will be displayed which contains the desired separator drawing. Note that the pixmap can only be set after the widget is created. If set when created, it will be overridden by the normal background pixmap created by the Primitive class.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

TRANSLATIONS

The menu separator widget defines no translations.

ACTIONS

The menu separator widget defines no actions.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X)

The **XtUnmanageChildren** function performs the following:

- Issues an error if the children do not all have the same parent or if the parent is not a subclass of **compositeWidgetClass.**
- Returns immediately if the common parent is being destroyed; otherwise, for each unique child on the list, **XtUnmanageChildren** performs the following:
  - Ignores the child if it already is unmanaged or is being destroyed and marks it if not.
  - If the child is realized, it makes it nonvisible by unmapping it.
- Calls the change_managed routine of the widgets' parent after all children have been marked if the parent is realized.

XtUnmanageChildren does not destroy the children widgets. Removing widgets from a parent's managed set is often a temporary banishment, and, some time later, you may manage the children again.

The **XtUnmanageChild** function constructs a widget list of length one and calls **XtUnmanageChildren.**

SEE ALSO

XtMapWidget(3Xt), XtRealizeWidget(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtNameToWidget, XtWidgetToWindow – translating strings to widgets or widgets to windows

SYNTAX

Widget XtNameToWidget(*reference, names*);
Widget *reference*;
String *names*;

Widget XtWindowToWidget(*display, window*)
Display **display*;
Window *window*;

ARGUMENTS

display	Specifies the display on which the window is defined.
names	Specifies the fully qualified name of the desired widget.
reference	Specifies the widget from which the search is to start.
window	Specify the window for which you want the widget.

DESCRIPTION

The **XtNameToWidget** function looks for a widget whose name is the first component in the specified names and that is a pop-up child of reference (or a normal child if reference is a subclass of **compositeWidgetClass**). It then uses that widget as the new reference and repeats the search after deleting the first component from the specified names. If it cannot find the specified widget, **XtNameToWidget** returns NULL.

Note that the names argument contains the name of a widget with respect to the specified reference widget and can contain more than one widget name (separated by periods) for widgets that are not direct children of the specified reference widget.

If more than one child of the reference widget matches the name, **XtNameToWidget** can return any of the children. The Intrinsics do not require that all children of a widget have unique names. If the specified names contain more than one component and if more than one child matches the first component, **XtNameToWidget** can return NULL if the single branch that it follows does not contain the named widget. That is, **XtNameToWidget** does not back up and follow other matching branches of the widget tree.

The **XtWindowToWidget** function translates the specified window and display pointer into the appropriate widget instance.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtOffset, XtNumber – determine the byte offset or number of array elements

SYNTAX

Cardinal XtOffset(*pointer_type, field_name*)
Type *pointer_type;*
Field *field_name;*

Cardinal XtNumber(*array*)
ArrayVariable *array;*

ARGUMENTS

array	Specifies a fixed-size array.
field_name	Specifies the name of the field for which to calculate the byte offset.
pointer_type	Specifies a type that is declared as a pointer to the structure.

DESCRIPTION

The **XtOffset** macro is usually used to determine the offset of various resource fields from the beginning of a widget and can be used at compile time in static initializations.

The **XtNumber** macro returns the number of elements in the specified argument lists, resources lists, and other counted arrays.

SEE ALSO

XtGetResourceList(3Xt), XtSetArg(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtOwnSelection, XtDisownSelection – set selection owner

SYNTAX

Boolean XtOwnSelection(*w, selection, time, convert_proc, lose_selection, done_proc*)
Widget *w;*
Atom *selection;*
Time *time;*
XtConvertSelectionProc *convert_proc;*
XtLoseSelectionProc *lose_selection;*
XtSelectionDoneProc *done_proc;*

void XtDisownSelection(*w, selection, time*)
Widget *w;*
Atom *selection;*
Time *time;*

ARGUMENTS

convert_proc	Specifies the procedure that is to be called whenever someone requests the current value of the selection.
done_proc	Specifies the procedure that is called after the requestor has received the selection or NULL if the owner is not interested in being called back.
lose_selection	Specifies the procedure that is to be called whenever the widget has lost selection ownership or NULL if the owner is not interested in being called back.
selection	Specifies an atom that describes the type of the selection (for example, XA_PRIMARY, XA_SECONDARY, or XA_CLIPBOARD).
time	Specifies the timestamp that indicates when the selection ownership should commence or is to be relinquished.
w	Specifies the widget that wishes to become the owner or to relinquish ownership.

DESCRIPTION

The **XtOwnSelection** function informs the Intrinsics selection mechanism that a widget believes it owns a selection. It returns **True** if the widget has successfully become the owner and **False** otherwise. The widget may fail to become the owner if some other widget has asserted ownership at a time later than this widget. Note that widgets can lose selection ownership either because someone else asserted later ownership of the selection or because the widget voluntarily gave up ownership of the selection. Also note that the lose_selection procedure is not called if the widget fails to obtain selection ownership in the first place.

The **XtDisownSelection** function informs the Intrinsics selection mechanism that the specified widget is to lose ownership of the selection. If the widget does not currently own the selection either because it lost the selection or because it never had the selection to begin with, **XtDisownSelection** does nothing.

After a widget has called **XtDisownSelection**, its convert procedure is not called even if a request arrives later with a timestamp during the period that this widget owned the selection. However, its done procedure will be called if a conversion that started before the call to **XtDisownSelection** finishes after the call to **XtDisownSelection**.

SEE ALSO

XtAppGetSelectionTimeout(3Xt), XtGetSelectionValue(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtPopup, XtCallbackNone, XtCallbackNonexclusive, XtCallbackExclusive, MenuPopup – map a pop-up

SYNTAX

void XtPopup(*popup_shell, grab_kind*)
Widget *popup_shell;*
XtGrabKind *grab_kind;*

void XtCallbackNone(*w, client_data, call_data*)
Widget *w;*
caddr_t *client_data;*
caddr_t *call_data;*

void XtCallbackNonexclusive(*w, client_data, call_data*)
Widget *w;*
caddr_t *client_data;*
caddr_t *call_data;*

void XtCallbackExclusive(*w, client_data, call_data*)
Widget *w;*
caddr_t *client_data;*
caddr_t *call_data;*

void MenuPopup(*shell_name*)
String *shell_name;*

ARGUMENTS

call_data	Specifies the callback data, which is not used by this procedure.
client_data	Specifies the pop-up shell.
grab_kind	Specifies the way in which user events should be constrained.
popup_shell	Specifies the widget shell.
w	Specifies the widget.

DESCRIPTION

The **XtPopup** function performs the following:

- Calls **XtCheckSubclass** to ensure popup_shell is a subclass of **Shell**.
- Generates an error if the shell's popped_up field is already **True**.
- Calls the callback procedures on the shell's popup_callback list.
- Sets the shell popped_up field to **True**, the shell spring_loaded field to **False**, and the shell grab_kind field from grab_kind.
- If the shell's create_popup_child field is non-NULL, **XtPopup** calls it with popup_shell as the parameter.
- If grab_kind is either **XtGrabNonexclusive** or **XtGrabExclusive**, it calls:

  XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), False)
- Calls **XtRealizeWidget** with popup_shell specified.
- Calls **XMapWindow** with popup_shell specified.

The **XtCallbackNone**, **XtCallbackNonexclusive**, and **XtCallbackExclusive** functions call **XtPopup** with the shell specified by the client data argument and grab_kind set as the name specifies. **XtCallbackNone**, **XtCallbackNonexclusive**, and **XtCallbackExclusive** specify **XtGrabNone**, **XtGrabNonexclusive**, and **XtGrabExclusive**, respectively. Each function then sets the widget that executed the callback list to be insensitive by using **XtSetSensitive**. Using these functions in callbacks is

not required. In particular, an application must provide customized code for callbacks that create pop-up shells dynamically or that must do more than desensitizing the button.

MenuPopup is known to the translation manager, which must perform special actions for spring-loaded pop-ups. Calls to **MenuPopup** in a translation specification are mapped into calls to a nonexported action procedure, and the translation manager fills in parameters based on the event specified on the left-hand side of a translation.

If **MenuPopup** is invoked on **ButtonPress** (possibly with modifiers), the translation manager pops up the shell with grab_kind set to **XtGrabExclusive** and spring_loaded set to **True**. If **MenuPopup** is invoked on **EnterWindow** (possibly with modifiers), the translation manager pops up the shell with grab_kind set to **XtGrabNonexclusive** and spring_loaded set to **False**. Otherwise, the translation manager generates an error. When the widget is popped up, the following actions occur:

- Calls **XtCheckSubclass** to ensure popup_shell is a subclass of **Shell**.
- Generates an error if the shell's popped_up field is already **True**.
- Calls the callback procedures on the shell's popup_callback list.
- Sets the shell popped_up field to **True** and the shell grab_kind and spring_loaded fields appropriately.
- If the shell's create_popup_child field is non-NULL, it is called with popup_shell as the parameter.
- Calls:

  XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), spring_loaded)
- Calls **XtRealizeWidget** with popup_shell specified.
- Calls **XMapWindow** with popup_shell specified.

(Note that these actions are the same as those for **XtPopup**.) **MenuPopup** tries to find the shell by searching the widget tree starting at the parent of the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops up the shell with the appropriate parameters. Otherwise, it moves up the parent chain as needed. If **MenuPopup** gets to the application widget and cannot find a matching shell, it generates an error.

SEE ALSO

XtCreatePopupShell(3Xt), XtPopdown(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtPopdown, XtCallbackPopdown, MenuPopdown – unmap a pop-up

SYNTAX

void XtPopdown(*popup_shell*)
Widget *popup_shell;*

void XtCallbackPopdown(*w, client_data, call_data*)
Widget *w;*
caddr_t *client_data;*
caddr_t *call_data;*

void MenuPopdown(*shell_name*)
String *shell_name;*

ARGUMENTS

call_data	Specifies the callback data, which is not used by this procedure.
client_data	Specifies a pointer to the **XtPopdownID** structure.
popup_shell	Specifies the widget shell to pop down.
shell_name	Specifies the name of the widget shell to pop down.
w	Specifies the widget.

DESCRIPTION

The **XtPopdown** function performs the following:

- Calls **XtCheckSubclass** to ensure popup_shell is a subclass of **Shell.**
- Checks that popup_shell is currently popped_up; otherwise, it generates an error.
- Unmaps popup_shell's window.
- If popup_shell's grab_kind is either **XtGrabNonexclusive** or **XtGrabExclusive,** it calls **XtRemoveGrab.**
- Sets pop-up shell's popped_up field to **False.**
- Calls the callback procedures on the shell's popdown_callback list.

The **XtCallbackPopdown** function casts the client data parameter to an **XtPopdownID** pointer:

typedef struct { Widget shell_widget; Widget enable_widget; } XtPopdownIDRec, *XtPopdownID; The shell_widget is the pop-up shell to pop down, and the enable_widget is the widget that was used to pop it up.

XtCallbackPopdown calls **XtPopdown** with the specified shell_widget and then calls **XtSetSensitive** to resensitize the enable_widget.

If a shell name is not given, **MenuPopdown** calls **XtPopdown** with the widget for which the translation is specified. If a shell_name is specified in the translation table, **MenuPopdown** tries to find the shell by looking up the widget tree starting at the parent of the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops down the shell; otherwise, it moves up the parent chain as needed. If **MenuPopdown** gets to the application top-level shell widget and cannot find a matching shell, it generates an error.

SEE ALSO

XtCreatePopupShell(3Xt), XtPopup(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtParseAcceleratorTable, XtInstallAccelerators, XtInstallAllAccelerators – managing accelerator tables

SYNTAX

XtAccelerators XtParseAcceleratorTable(*source*)
String *source;*

void XtInstallAccelerators(*destination, source*)
Widget *destination;*
Widget *source;*

void XtInstallAllAccelerators(*destination, source*)
Widget *destination;*
Widget *source;*

ARGUMENTS

source	Specifies the accelerator table to compile.
destination	Specifies the widget on which the accelerators are to be installed.
source	Specifies the widget or the root widget of the widget tree from which the accelerators are to come.

DESCRIPTION

The **XtParseAcceleratorTable** function compiles the accelerator table into the opaque internal representation.

The **XtInstallAccelerators** function installs the accelerators from source onto destination by augmenting the destination translations with the source accelerators. If the source display_accelerator method is non-NULL, **XtInstallAccelerators** calls it with the source widget and a string representation of the accelerator table, which indicates that its accelerators have been installed and that it should display them appropriately. The string representation of the accelerator table is its canonical translation table representation.

The **XtInstallAllAccelerators** function recursively descends the widget tree rooted at source and installs the accelerators of each widget encountered onto destination. A common use os to call **XtInstallAllAccelerators** and pass the application main window as the source.

SEE ALSO

XtParseTranslationTable(1)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtParseTranslationTable, XtAugmentTranslations, XtOverrideTranslations, XtUninstallTranslations – manage translation tables

SYNTAX

XtTranslations XtParseTranslationTable(*table*)
String *table;*

void XtAugmentTranslations(*w, translations*)
Widget *w;*
XtTranslations *translations;*

void XtOverrideTranslations(*w, translations*)
Widget *w;*
XtTranslations *translations;*

void XtUninstallTranslations(*w*)
Widget *w;*

ARGUMENTS

table	Specifies the translation table to compile.
translations	Specifies the compiled translation table to merge in (must not be NULL).
w	Specifies the widget into which the new translations are to be merged or removed.

DESCRIPTION

The **XtParseTranslationTable** function compiles the translation table into the opaque internal representation of type **XtTranslations**. Note that if an empty translation table is required for any purpose, one can be obtained by calling **XtParseTranslationTable** and passing an empty string.

The **XtAugmentTranslations** function nondestructively merges the new translations into the existing widget translations. If the new translations contain an event or event sequence that already exists in the widget's translations, the new translation is ignored.

The **XtOverrideTranslations** function destructively merges the new translations into the existing widget translations. If the new translations contain an event or event sequence that already exists in the widget's translations, the new translation is merged in and override the widget's translation.

To replace a widget's translations completely, use **XtSetValues** on the XtNtranslations resource and specifiy a compiled translation table as the value.

The **XtUninstallTranslations** function causes the entire translation table for widget to be removed.

SEE ALSO

XtAppAddActions(3Xt), XtCreatePopupShell(3Xt), XtParseAcceleratorTable(3Xt), XtPopup(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtQueryGeometry – query the preferred geometry of a child widget

SYNTAX

XtGeometryResult XtQueryGeometry(*w, intended, preferred_return*)
Widget *w*;
XtWidgetGeometry **intended*, **preferred_return*;

ARGUMENTS

intended	Specifies any changes the parent plans to make to the child's geometry or NULL.
preferred_return	Returns the child widget's preferred geometry.
w	Specifies the widget.

DESCRIPTION

To discover a child's preferred geometry, the child's parent sets any changes that it intends to make to the child's geometry in the corresponding fields of the intended structure, sets the corresponding bits in intended.request_mode, and calls **XtQueryGeometry**.

XtQueryGeometry clears all bits in the preferred_return->request_mode and checks the query_geometry field of the specified widget's class record. If query_geometry is not NULL, **XtQueryGeometry** calls the query_geometry procedure and passes as arguments the specified widget, intended, and preferred_return structures. If the intended argument is NULL, **XtQueryGeometry** replaces it with a pointer to an **XtWidgetGeometry** structure with request_mode=0 before calling query_geometry.

SEE ALSO

XtConfigureWidget(3Xt), XtMakeGeometryRequest(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtRealizeWidget, XtIsRealized, XtUnrealizeWidget – realize and unrealize widgets

SYNTAX

void XtRealizeWidget(*w*)
Widget *w*;

Boolean XtIsRealized(*w*)
Widget *w*;

void XtUnrealizeWidget(*w*)
Widget *w*;

ARGUMENTS

w Specifies the widget.

DESCRIPTION

If the widget is already realized, **XtRealizeWidget** simply returns. Otherwise, it performs the following:

- Binds all action names in the widget's translation table to procedures (see Section 10.1.2).
- Makes a post-order traversal of the widget tree rooted at the specified widget and calls the change_managed procedure of each composite widget that has one or more managed children.
- Constructs an **XSetWindowAttributes** structure filled in with information derived from the **Core** widget fields and calls the realize procedure for the widget, which adds any widget-specific attributes and creates the X window.
- If the widget is not a subclass of **compositeWidgetClass**, **XtRealizeWidget** returns; otherwise, it continues and performs the following:
  - Descends recursively to each of the widget's managed children and calls the realize procedures. Primitive widgets that instantiate children are responsible for realizing those children themselves.
  - Maps all of the managed children windows that have mapped_when_managed **True**. (If a widget is managed but mapped_when_managed is **False**, the widget is allocated visual space but is not displayed. Some people seem to like this to indicate certain states.)

If the widget is a top-level shell widget (that is, it has no parent), and mapped_when_managed is **True**, **XtRealizeWidget** maps the widget window.

The **XtIsRealized** function returns **True** if the widget has been realized, that is, if the widget has a nonzero X window ID.

Some widget procedures (for example, set_values) might wish to operate differently after the widget has been realized.

The **XtUnrealizeWidget** function destroys the windows of an existing widget and all of its children (recursively down the widget tree). To recreate the windows at a later time, call **XtRealizeWidget** again. If the widget was managed, it will be unmanaged automatically before its window is freed.

SEE ALSO

XtManageChildren(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtSetArg, XtMergeArgLists – set and merge ArgLists

SYNTAX

XtSetArg(*arg, name, value*)
Arg *arg;*
String *name;*
XtArgVal *value;*

ArgList XtMergeArgLists(*args1, num_args1, args2, num_args2*)
ArgList *args1;*
Cardinal *num_args1;*
ArgList *args2;*
Cardinal *num_args2;*

ARGUMENTS

arg	Specifies the name-value pair to set.
args1	Specifies the first **ArgList**.
args2	Specifies the second **ArgList**.
num_args1	Specifies the number of arguments in the first argument list.
num_args2	Specifies the number of arguments in the second argument list.
name	Specifies the name of the resource.
value	Specifies the value of the resource if it will fit in an **XtArgVal** or the address.

DESCRIPTION

The **XtSetArg** function is usually used in a highly stylized manner to minimize the probability of making a mistake; for example:

```
Arg args[20]; int n;

n = 0; XtSetArg(args[n], XtNheight, 100);          n++; XtSetArg(args[n],
XtNwidth, 200);          n++; XtSetValues(widget, args, n);
```

Alternatively, an application can statically declare the argument list and use **XtNumber**:

```
static Args args[] = {     {XtNheight, (XtArgVal) 100},     {XtNwidth,
(XtArgVal) 200}, }; XtSetValues(Widget, args, XtNumber(args));
```

Note that you should not use auto-increment or auto-decrement within the first argument to **XtSetArg**. **XtSetArg** can be implemented as a macro that dereferences the first argument twice.

The **XtMergeArgLists** function allocates enough storage to hold the combined **ArgList** structures and copies them into it. Note that it does not check for duplicate entries. When it is no longer needed, free the returned storage by using **XtFree**.

SEE ALSO

XtOffset(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtSetKeyboardFocus – focus events on a child widget

SYNTAX

XtSetKeyboardFocus(*subtree, descendant*)
Widget *subtree, descendant;*

ARGUMENTS

descendant	Specifies either the widget in the subtree structure which is to receive the keyboard event, or **None**. Note that it is not an error to specify **None** when no input focus was previously set.
w	Specifies the widget for which the keyboard focus is to be set.

DESCRIPTION

If a future **KeyPress** or **KeyRelease** event occurs within the specified subtree, **XtSetKeyboardFocus** causes **XtDispatchEvent** to remap and send the event to the specified descendant widget.

When there is no modal cascade, keyboard events can occur within a widget W in one of three ways:

- W has the X input focus.
- W has the keyboard focus of one of its ancestors, and the event occurs within the ancestor or one of the ancestor's descendants.
- No ancestor of W has a descendant within the keyboard focus, and the pointer is within W.

When there is a modal cascade, a widget W receives keyboard events if an ancestor of W is in the active subset of the modal cascade and one or more of the previous conditions is **True**.

When subtree or one of its descendants acquires the X input focus or the pointer moves into the subtree such that keyboard events would now be delivered to subtree, a **FocusIn** event is generated for the descendant if **FocusNotify** events have been selected by the descendant. Similarly, when W loses the X input focus or the keyboard focus for one of its ancestors, a **FocusOut** event is generated for descendant if **FocusNotify** events have been selected by the descendant.

SEE ALSO

XtCallAcceptFocus(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtSetKeyTranslator, XtTranslateKeycode, XtRegisterCaseConverter, XtConvertCase – convert KeySym to KeyCodes

SYNTAX

void XtSetKeyTranslator(*display, proc*)
Display **display*;
XtKeyProc *proc*;

void XtTranslateKeycode(*display, keycode, modifiers, modifiers_return, keysym_return*)
Display **display*;
KeyCode *keycode*;
Modifiers *modifiers*;
Modifiers **modifiers_return*;
KeySym **keysym_return*;

void XtRegisterCaseConverter(*display, proc, start, stop*)
Display **display*;
XtCaseProc *proc*;
KeySym *start*;
KeySym *stop*;

void XtConvertCase(*display, keysym, lower_return, upper_return*)
Display **display*;
KeySym *keysym*;
KeySym **lower_return*;
KeySym **upper_return*;

ARGUMENTS

display	Specifies the display.
keycode	Specifies the KeyCode to translate.
keysym	Specifies the KeySym to convert.
keysym_return	Returns the resulting KeySym.
lower_return	Returns the lowercase equivalent of the KeySym.
upper_return	Returns the uppercase equivalent of the KeySym.
modifiers	Specifies the modifiers to the KeyCode.
modifiers_return	Returns a mask that indicates the modifiers actually used to generate the KeySym.
proc	Specifies the procedure that is to perform key translations or conversions.
start	Specifies the first KeySym for which this converter is valid.
stop	Specifies the last KeySym for which this converter is valid.

DESCRIPTION

The **XtSetKeyTranslator** function sets the specified procedure as the current key translator. The default translator is **XtTranslateKey**, an **XtKeyProc** that uses Shift and Lock modifiers with the interpretations defined by the core protocol. It is provided so that new translators can call it to get default KeyCode-to-KeySym translations and so that the default translator can be reinstalled.

The **XtTranslateKeycode** function passes the specified arguments directly to the currently registered KeyCode to KeySym translator.

The **XtRegisterCaseConverter** registers the specified case converter. The start and stop arguments provide the inclusive range of KeySyms for which this converter is to be called. The new converter overrides any previous converters for KeySyms in that range. No interface exists to remove converters; you need to register an identity converter. When a new converter is registered, the Intrinsics refreshes the keyboard state if necessary. The default converter understands case conversion for all KeySyms defined in the core protocol.

The **XtConvertCase** function calls the appropriate converter and returns the results. A user-supplied **XtKeyProc** may need to use this function.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtSetSensitive, XtIsSensitive – set and check a widget's sensitivity state

SYNTAX

void XtSetSensitive(*w*, *sensitive*)
Widget *w*;
Boolean *sensitive*;

Boolean XtIsSensitive(*w*)
Widget *w*;

ARGUMENTS

sensitive	Specifies a Boolean value that indicates whether the widget should receive keyboard and pointer events.
w	Specifies the widget.

DESCRIPTION

The **XtSetSensitive** function first calls **XtSetValues** on the current widget with an argument list specifying that the sensitive field should change to the new value. It then recursively propagates the new value down the managed children tree by calling **XtSetValues** on each child to set the ancestor_sensitive to the new value if the new values for sensitive and the child's ancestor_sensitive are not the same.

XtSetSensitive calls **XtSetValues** to change sensitive and ancestor_sensitive. Therefore, when one of these changes, the widget's set_values procedure should take whatever display actions are needed (for example, greying out or stippling the widget).

XtSetSensitive maintains the invariant that if parent has either sensitive or ancestor_sensitive **False**, then all children have ancestor_sensitive **False**.

The **XtIsSensitive** function returns **True** or **False** to indicate whether or not user input events are being dispatched. If both core.sensitive and core.ancestor_sensitive are **True**, **XtIsSensitive** returns **True**; otherwise, it returns **False**.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtSetValues, XtSetSubvalues, XtGetValues, XtGetSubvalues – obtain and set widget resources

SYNTAX

void XtSetValues(*w, args, num_args*)
Widget *w;*
ArgList *args;*
Cardinal *num_args;*

void XtSetSubvalues(*base, resources, num_resources, args, num_args*)
caddr_t *base;*
XtResourceList *resources;*
Cardinal *num_resources;*
ArgList *args;*
Cardinal *num_args;*

void XtGetValues(*w, args, num_args*)
Widget *w;*
ArgList *args;*
Cardinal *num_args;*

void XtGetSubvalues(*base, resources, num_resources, args, num_args*)
caddr_t *base;*
XtResourceList *resources;*
Cardinal *num_resources;*
ArgList *args;*
Cardinal *num_args;*

ARGUMENTS

args	Specifies the argument list of name/address pairs that contain the resource name and either the address into which the resource value is to be stored or their new values.
base	Specifies the base address of the subpart data structure where the resources should be retrieved or written.
num_args	Specifies the number of arguments in the argument list.
resources	Specifies the nonwidget resource list or values.
num_resources	Specifies the number of resources in the resource list.
w	Specifies the widget.

DESCRIPTION

The **XtSetValues** function starts with the resources specified for the **Core** widget fields and proceeds down the subclass chain to the widget. At each stage, it writes the new value (if specified by one of the arguments) or the existing value (if no new value is specified) to a new widget data record. **XtSetValues** then calls the set_values procedures for the widget in superclass-to-subclass order. If the widget has any non-NULL set_values_hook fields, these are called immediately after the corresponding set_values procedure. This procedure permits subclasses to set nonwidget data for **XtSetValues.**

If the widget's parent is a subclass of **constraintWidgetClass, XtSetValues** also updates the widget's constraints. It starts with the constraint resources specified for **constraintWidgetClass** and proceeds down the subclass chain to the parent's class. At each stage, it writes the new value or the existing value to a new constraint record. It then calls the constraint set_values procedures from **constraintWidgetClass** down to the parent's class. The constraint set_values procedures are called with widget

arguments, as for all set_values procedures, not just the constraint record arguments, so that they can make adjustments to the desired values based on full information about the widget.

XtSetValues determines if a geometry request is needed by comparing the current widget to the new widget. If any geometry changes are required, it makes the request, and the geometry manager returns **XtGeometryYes**, **XtGeometryAlmost**, or **XtGeometryNo**. If **XtGeometryYes**, **XtSetValues** calls the widget's resize procedure. If **XtGeometryNo**, **XtSetValues** resets the geometry fields to their original values. If **XtGeometryAlmost**, **XtSetValues** calls the set_values_almost procedure, which determines what should be done and writes new values for the geometry fields into the new widget. **XtSetValues** then repeats this process, deciding once more whether the geometry manager should be called.

Finally, if any of the set_values procedures returned **True**, **XtSetValues** causes the widget's expose procedure to be invoked by calling the Xlib **XClearArea** function on the widget's window.

The **XtSetSubvalues** function stores resources into the structure identified by base.

The **XtGetValues** function starts with the resources specified for the core widget fields and proceeds down the subclass chain to the widget. The value field of a passed argument list should contain the address into which to store the corresponding resource value. It is the caller's responsibility to allocate and deallocate this storage according to the size of the resource representation type used within the widget.

If the widget's parent is a subclass of **constraintWidgetClass**, **XtGetValues** then fetches the values for any constraint resources requested. It starts with the constraint resources specified for **constraintWidgetClass** and proceeds down to the subclass chain to the parent's constraint resources. If the argument list contains a resource name that is not found in any of the resource lists searched, the value at the corresponding address is not modified. Finally, if the get_values_hook procedures are non-NULL, they are called in superclass-to-subclass order after all the resource values have been fetched by **XtGetValues**. This permits a subclass to provide nonwidget resource data to **XtGetValues**.

The **XtGetSubvalues** function obtains resource values from the structure identified by base.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

XtStringConversionWarning (3Xt)

NAME

XtStringConversionWarning – issue a conversion warning message

SYNTAX

void XtStringConversionWarning(*src*, *dst_type*)
String *src*, *dst_type*;

ARGUMENTS

src	Specifies the string that could not be converted.
dst_type	Specifies the name of the type to which the string could not be converted.

DESCRIPTION

The **XtStringConversionWarning** function issues a warning message with name "conversionError", type "string", class "XtToolkitError, and the default message string "Cannot convert "*src*" to type *dst_type*".

SEE ALSO

XtAppAddConverter(3Xt), XtAppErrorMsg(3t), XtConvert(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtTranslateCoordinates – translate widget coordinates

SYNTAX

void XtTranslateCoords(*w, x, y, rootx_return, rooty_return*)
Widget *w;*
Position *x, y;*
Position **rootx_return, *rooty_return;*

ARGUMENTS

rootx_return	
rooty_return	Returns the root-relative x and y coordinates.
x	
y	Specify the widget-relative x and y coordinates.
w	Specifies the widget.

DESCRIPTION

While **XtTranslateCoords** is similar to the Xlib **XTranslateCoordinates** function, it does not generate a server request because all the required information already is in the widget's data structures.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

WIDGETS *MAN* PAGES

CHAPTER THREE

NAME

XtAddEventHandler, XtAddRawEventHandler, XtRemoveEventHandler XtRemoveRawEventHandler – add and remove event handlers

SYNTAX

void XtAddEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w;*
EventMask *event_mask;*
Boolean *nonmaskable;*
XtEventHandler *proc;*
caddr_t *client_data;*

void XtAddRawEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w;*
EventMask *event_mask;*
Boolean *nonmaskable;*
XtEventHandler *proc;*
caddr_t *client_data;*

void XtRemoveEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w;*
EventMask *event_mask;*
Boolean *nonmaskable;*
XtEventHandler *proc;*
caddr_t *client_data;*

void XtRemoveRawEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w;*
EventMask *event_mask;*
Boolean *nonmaskable;*
XtEventHandler *proc;*
caddr_t *client_data;*

ARGUMENTS

client_data	Specifies additional data to be passed to the client's event handler.
event_mask	Specifies the event mask for which to call or unregister this procedure.
nonmaskable	Specifies a Boolean value that indicates whether this procedure should be called or removed on the nonmaskable events (**GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage,** and **MappingNotify**).
proc	Specifies the procedure that is to be added or removed.
w	Specifies the widget for which this event handler is being registered.

DESCRIPTION

The **XtAddEventHandler** function registers a procedure with the dispatch mechanism that is to be called when an event that matches the mask occurs on the specified widget. If the procedure is already registered with the same client_data, the specified mask is ORed into the existing mask. If the widget is realized, **XtAddEventHandler** calls **XSelectInput**, if necessary.

The **XtAddRawEventHandler** function is similar to **XtAddEventHandler** except that it does not affect the widget's mask and never causes an **XSelectInput** for its events. Note that the widget might already have those mask bits set because of other nonraw event handlers registered on it.

The **XtAddRawEventHandler** function is similar to **XtAddEventHandler** except that it does not affect the widget's mask and never causes an **XSelectInput** for its events. Note that the widget might already have those mask bits set because of other nonraw event handlers registered on it.

The **XtRemoveRawEventHandler** function stops the specified procedure from receiving the specified events. Because the procedure is a raw event handler, this does not affect the widget's mask and never causes a call on **XSelectInput**.

SEE ALSO

XtAppNextEvent(3Xt), XtBuildEventMask(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

XtAddExposureToRegion (3Xt)

NAME

XtAddExposureToRegion – merge exposure events into a region

SYNTAX

void XtAddExposureToRegion(*event, region*)
XEvent **event*;
Region *region*;

ARGUMENTS

event	Specifies a pointer to the **Expose** or **GraphicsExpose** event.
region	Specifies the region object (as defined in **<X11/Xutil.h>**).

DESCRIPTION

The **XtAddExposureToRegion** function computes the union of the rectangle defined by the exposure event and the specified region. Then, it stores the results back in region. If the event argument is not an **Expose** or **GraphicsExpose** event, **XtAddExposureToRegion** returns without an error and without modifying region.

This function is used by the exposure compression mechanism (see Section 7.9.3).

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtAddCallback, XtAddCallbacks, XtRemoveCallback, XtRemoveCallbacks, XtRemoveAllCallbacks – add and remove callback procedures

SYNTAX

void XtAddCallback(*w, callback_name, callback, client_data*)
Widget *w;*
String *callback_name;*
XtCallbackProc *callback;*
caddr_t *client_data;*

void XtAddCallbacks(*w, callback_name, callbacks*)
Widget *w;*
String *callback_name;*
XtCallbackList *callbacks;*

void XtRemoveCallback(*w, callback_name, callback, client_data*)
Widget *w;*
String *callback_name;*
XtCallbackProc *callback;*
caddr_t *client_data;*

void XtRemoveCallbacks(*w, callback_name, callbacks*)
Widget *w;*
String *callback_name;*
XtCallbackList *callbacks;*

void XtRemoveAllCallbacks(*w, callback_name*)
Widget *w;*
String *callback_name;*

ARGUMENTS

callback	Specifies the callback procedure.
callbacks	Specifies the null-terminated list of callback procedures and corresponding client data.
callback_name	Specifies the callback list to which the procedure is to be appended or deleted.
client_data	Specifies the argument that is to be passed to the specified procedure when it is invoked by XtCallbacks or NULL, or the client data to match on the registered callback procedures.
w	Specifies the widget.

DESCRIPTION

The **XtAddCallback** function adds the specified callback procedure to the specified widget's callback list.

The **XtAddCallbacks** add the specified list of callbacks to the specified widget's callback list.

The **XtRemoveCallback** function removes a callback only if both the procedure and the client data match.

The **XtRemoveCallbacks** function removes the specified callback procedures from the specified widget's callback list.

The **XtRemoveAllCallbacks** function removes all the callback procedures from the specified widget's callback list.

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	10
XtNheight	XtCHeight	int	10
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalOn	XtCTraversalOn	Boolean	TRUE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at create time or during a call to XtSetValues, the XwManager superclass will automatically augment the bulletin board manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X)

NAME

XwarrowWidgetClass – the X Widget's arrow drawing widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Arrow.h>
```

CLASSES

The Arrow widget is built from the Core and XwPrimitive classes.

The widget class to use when creating an arrow is **XwarrowWidgetClass.** The class name for this widget is **Arrow.**

DESCRIPTION

The Arrow widget supports drawing of an arrow within the bounds of its window. It uses the primitive widget's border highlighting routines.

The arrow can be drawn in the directions of up, down, left and right. The Arrow widget also supports two types of callbacks: Button selections, and Button releases.

NEW RESOURCES

The Arrow widget defines a set of resources used by the programmer to specify the data for the arrow. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by the Arrow widget.

Arrow Resource Set			
Name	Class	Type	Default
XtNarrowDirection	XtCArrowDirection	int	up

XtNarrowDirection

This resource is the means by which the arrow direction is set. It can be defined in either of two ways: Through the .Xdefaults file by the strings "up", "down", "left" and "right". Within an arg list for use in XtSetValues() by the defines XwARROW_UP, XwARROW_DOWN, XwARROW_LEFT and XwARROW_RIGHT.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

XWARROW(3Xh)

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

Input to the Arrow widget is driven by the mouse buttons. The Primitive class resources of XtNselect and XtNrelease define the callback lists used by the Arrow widget. Thus, to receive input from an arrow, the application adds callbacks to the arrow using these two resource types. The default translation set for the Arrow widget is as follows.

<Btn1Down>:	select()	
<Btn1Up>:	release()	
<EnterWindow>:	enter()	
<LeaveWindow>:	leave()	
<KeyDown>Select:	select()	HP "Select" key
<KeyUp>Select:	unselect()	HP "Select" key

ACTIONS

select:
Selections occurring on an arrow cause the arrow to be displayed as selected and its primitive XtNselect callbacks are called.

release:
Release redraws the arrow in its normal mode and calls its primitive XtNrelease callbacks.

enter:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be highlighted. Otherwise no action is taken.

leave:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWCREATETILE(3X)

NAME

XwbulletinWidgetClass – the X Widgets bulletin board manager widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/BBoard.h>
```

CLASSES

The bulletin board manager widget is built from the Core, Composite, Constraint and XwManager classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no resources that the user can set, there is no table for Composite class resources.

The widget class to use when creating a bulletin board is **XwbulletinWidgetClass.** The class name is **BulletinBoard.**

DESCRIPTION

The bulletin board manager widget is a composite widget that enforces no ordering on its children. It is up to the application to specify the x and y coordinates of the children inserted into this widget, otherwise they will all appear at (0,0).

This manager widget supports 3 different layout policies: minimize (the default), maximize and ignore. When the layout policy is set to minimize, the manager will create a box that is just large enough to contain all of its children, regardless of any provided width and height values. The ignore setting forces the manager to honor its given width and height, it will not grow or shrink in response to the addition, deletion or altering of its children. When set to the maximize setting, the BulletinBoard widget will ask for additional space when it needs it, but will not give up extra space.

The bulletin board manager also implements the X Widgets keyboard interface.

No callbacks are defined for this manager.

NEW RESOURCES

The bulletin board manager widget class does not define any additional resources; all necessary resources are present in its superclasses. The programmer should refer to the man pages for the bulletin board's superclasses to determine the resources that can be set and the defaults settings for these resources.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	10
XtNheight	XtCHeight	int	10
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalOn	XtCTraversalOn	Boolean	TRUE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at create time or during a call to XtSetValues, the XwManager superclass will automatically augment the bulletin board manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X)

NAME

XwbuttonWidgetClass – X Widget Button MetaClass

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
```

CLASSES

The XwButtonClass is built from the Core and XwPrimitive classes.

DESCRIPTION

The XwButton class is an X Widget meta class. It is never instantiated as a widget. It provides a set of resources that are needed by a variety of other X Widgets (for example: XwtoggleWidgetClass and XwpushButtonWidgetClass).

NEW RESOURCES

The XwButtonClass defines a set of resource types used by the programmer to specify the data for widgets that are subclasses of XwButtonClass. To specify any of these resources within the .Xdefaults file, drop the XtN prefix from the resource name. For example, XtNfont becomes font.

Button Resource Set			
Name	**Class**	**Type**	**Default**
XtNfont	XtCFont	XFontStruct *	Fixed
XtNlabel	XtCLabel	caddr_t	NULL
XtNlabelLocation	XtCLabelLocation	int	right
XtNvSpace	XtCVSpace	int	2
XtNhSpace	XtCHSpace	int	2
XtNset	XtCSet	Boolean	FALSE
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground
XtNborderWidth	XtCBorderWidth	int	0

XtNfont
The application may define the font to be used when displaying the button string. Any valid X11 font may be used.

XtNlabel
The application may define the button label by providing a pointer to a null terminated character string. If no label is provided the class name of the widget will be used.

XtNlabelLocation
For those buttons that have a separate graphic, this field specifies whether the label should appear to the left or to the right of that graphic. The acceptable values are the defines XwRIGHT (the default) and XwLEFT.

XtNvSpace
The application may determine the number of pixels of space left between the top of the button and the top of the button label, and between the bottom of the label and the bottom of the button.

XtNhSpace
The application may determine the number of pixels of space left between the left side of the button and the leftmost part of the button label, and between the rightmost part of the button label and the right side of the button.

XtNset
If set to true the button will display itself in its selected state. This is useful for showing some conditions as active when a set of buttons appear.

XtNsensitiveTile
The application can determine the mix of foreground and background that will be used to draw text to show insensitivity. The #defines for setting the values through an arg list and the strings to be used in the .Xdefault file are described in XwCreateTile(3X). The default is Xw75_FOREGROUND which is a 75/25 mix of foreground and background colors.

XtNborderWidth
This redefines the core class default border width from 1 pixel to 0 pixels.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

XWPRIMITIVE(3X)

NAME

XwcascadeWidgetClass – the X Widgets popup and pulldown menupane widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/Cascade.h>
```

CLASSES

The Cascade menupane widget is built from the Core, Composite, Constraint, XwManager and XwMenuPane classes. Note that the Constraint fields are not used in this widget and are not listed in the resource tables below. Also, since the Composite class contains no resources that can be set by the user, there is no table for Composite class resources.

The widget class to use when creating a cascading menupane is **XwcascadeWidgetClass**. The class name is **Cascade**.

DESCRIPTION

The Cascade menupane widget is a composite widget which may be used by an application when creating a set of menus.

The Cascade menupane widget always displays its managed children in a single column, and always attempts to size itself to the smallest possible size, as described by the children it contains; as the children grow or shrink in size, the menupane will attempt to adapt its size accordingly.

The Cascade menupane widget allows a title to be displayed at the top of the menupane, the bottom of the menupane, or at both places. Additionally, the title may be either a text string or an image. The title is always centered horizontally within the menupane.

Refer to the manual page for XwManager(3X) for a description of how to specify the order in which menubuttons are inserted into a menupane.

NEW RESOURCES

The MenuPane defines a set of resource types used by the programmer to specify the data for the menupane. The programmer can also set the values for the Core, Composite Manager and MenuPane widget classes to set attributes for this widget. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by Cascade.

Cascade Resource Set			
Name	Class	Type	Default
XtNtitlePosition	XtCTitlePosition	int	top

XtNtitlePosition

This resource is used to control where the title is displayed within the cascading menupane. To programmatically set this resource, use either the XwTOP, XwBOTTOM or XwBOTH define. To set this resource using the .Xdefaults file, use one of the strings top, bottom or both.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

MenuPane Resource Set -- XWMENUPANE(3X)			
Name	**Class**	**Type**	**Default**
XtNtitleShowing	XtCTitleShowing	Boolean	FALSE
XtNmgrTitleOverride	XtCTitleOverride	Boolean	FALSE
XtNtitleType	XtCTitleType	int	XwSTRING
XtNtitleString	XtCTitleString	String	widget name
XtNtitleImage	XtCTitleImage	XImage *	NULL
XtNfont	XtCFont	XFontStruct *	"fixed"
XtNattachTo	XtCAttachTo	String	NULL
XtNmnemonic	XtCMnemonic	String	NULL
XtNselect	XtCCallback	Pointer	NULL

TRANSLATIONS

The input to the Cascade menupane widget is driven by the mouse buttons. The default translations set by this widget are as follows:

<Btn1Down>:	select()
<LeaveWindow>:	leave()
<visible>:	visible()
<unmap>:	unmap()

ACTIONS

select:
Informs the menu manager, if present, that a select occurred, and then invokes the select callbacks, unless instructed not to by the menu manager. If no menu manager is present, then the select callbacks will be invoked.

leave:
This routine overrides the leave action routine provided by the XwManager meta class.

visible:
This action overrides the visible action routine provided by the XwManager meta class.

unmap:
This action overrides the unmap action provided by the XwManager meta class.

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X), XWMENUPANE(3X)

NAME

XwCreateTile – create a tile suitable for area filling or patterned text.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

Pixmap XwCreateTile (screen, foreground, background, tileType)
Screen * screen;
Pixel foreground;
Pixel background;
int tileType;
```

ARGUMENTS

screen
This parameter contains the screen for which the tile is to be created.

foreground
This is the foreground color to use for creating the tile.

background
This is the background color to use for creating the tile.

tileType
This is an integer value representing a particular pattern to use when creating the tile.

DESCRIPTION

XwCreateTile is a function (not a widget) that creates and returns a pixmap of screen depth, using the foreground and background colors specified. The tileType parameter is used to select the particular tile to create. Duplicate requests for the same tile, screen, foreground and background are cached to reduce overhead.

There are nine available tile types. They are defined by a set of #define statements in the file Xw.h and are described in the following table.

Define	Description
XwFOREGROUND	A tile of solid foreground
XwBACKGROUND	A tile of solid background
Xw25_FOREGROUND	A tile of 25% foreground, 75% background
Xw50_FOREGROUND	A tile of 50% foreground, 50% background
Xw75_FOREGROUND	A tile of 75% foreground, 25% background
XwHORIZONTAL_TILE	A tile of horizontal lines of the two colors
XwVERTICAL_TILE	A tile of vertical lines of the two colors
XwSLANT_RIGHT	A tile of slanting lines of the two colors
XwSLANT_LEFT	A tile of slanting lines of the two colors

To use a tile created by this function, the returned tile should be placed into the tile field of a graphics context, and the fill_style should be set to FillTiled.

RESOURCES

XwCreateTile gives the application or widget writer an easy mechanism to specify the tile type to use. The tile type can be specified within the .Xdefaults file or an argument list. A resource converter is present to convert .Xdefault strings into the matching defined value for each of the tiles. The strings to be contained within the .Xdefaults file are as follows.

Xdefault String	Define
foreground	XwFOREGROUND
background	XwBACKGROUND
25_foreground	Xw25_FOREGROUND
50_foreground	Xw50_FOREGROUND
75_foreground	Xw75_FOREGROUND
horizontal_tile	XwHORIZONTAL_TILE
vertical_tile	XwVERTICAL_TILE
slant_right	XwSLANT_RIGHT
slant_left	XwSLANT_LEFT

For widget writers who wish to incorporate settable tiles within their resource set, the representation member of the resource definition should be set to the define XtRTileType.

RETURN VALUES

XwCREATETILE returns a pixmap if successful. If an invalid tile type or screen is specified, 0 is returned.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

NAME

XwformWidgetClass – the X Widget's general widget layout manager

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Form.h>

CLASSES

A Form widget is built from Core, Composite, Constraint and XwManager classes

The widget class to use when creating a form is **XwformWidgetClass.**

The class name of Form is **Form.**

DESCRIPTION

The Form widget is a constraint based manager that provides a layout language used to establish spatial relationships between its children and then manipulates these relationships when the Form is resized, new children are added to the Form, or its children are resized, unmanaged, remanaged or destroyed.

The following list highlights the types of layout control provided by the form widget.

Spanning Constraints
A widget can be created with a set of constraints such that it spans the width or height of a form. This is often used for the layout of scrollbars and titlebars. Constraints that cause a widget to span both the width and height of a form can also be specified.

Row Constraints
Sets of widgets can be set up as a row so that resizing a form may increase or decrease the spacing between the widgets. The form may also make the widgets smaller if desired, but it will not allow the widgets to overlap.

Column Constraints
Sets of widgets can be displayed in a single column or in multiple columns. The form may increase or decrease the spacing between widgets or resize the widgets, but it will not allow the widgets to overlap.

Automatic Form Resizing
The form calculates new sizes or positions for its children whenever they change size or position. The new orm size thus generated is passed as a *geometry request* to the parent of the form. The parent can accept the request or modify it and return it as a *geometry almost.* When a geometry almost is returned by the parent, the form respecifies the constraints to match the parent's reply size.

Optimal Child Sizes and Positions
The Form widget also calculates the sizes and positions of its children to both match the constraints defined and to match either the initial size of the widget or the size given when the widget was modified through XtSetValues. These values are further constrained to match a given form size only when the form's size is being explicitly changed through its resize procedure, or its parent returns a geometry almost when the form makes a geometry request.

Managing, Unmanaging and Destroying Children
When a widget within a form is unmanaged or destroyed, it is removed from the constraint processing and the constraints are reprocessed to reposition and/or resize the form and its contents. Any widgets that referenced it are

rereferenced to the widget that it had been referencing. For the unmanaged case, if the widget is remanaged, the widgets that were previously referencing it are rereferenced to it, thus preserving the original layout.

NEW RESOURCES

The Form does not add any new resources. All of the functionality for the form is tied to its constraint resources.

CONSTRAINT RESOURCES

The following resources are attached to every widget inserted into Form. To specify an of these resources within a .Xdefaults file, drop the XtN from the resource name. Refer to CONSTRAINT(3X) for a general discussion of constraint resources.

Constraint Resource Set -- Children of FORM(3X)			
Name	**Class**	**Type**	**Default**
XtNxRefName	XtCXRefName	String	NULL
XtNxRefWidget	XtCXRefWidget	Widget	the parent form
XtNxOffset	XtCXOffset	int	0
XtNxAddWidth	XtCXAddWidth	Boolean	FALSE
XtNxVaryOffset	XtCXVaryOffset	Boolean	FALSE
XtNxResizable	XtCXResizable	Boolean	FALSE
XtNxAttachRight	XtCXAttachRight	Boolean	FALSE
XtNxAttachOffset	XtCXAttachOffset	int	0
XtNyRefName	XtCYRefName	String	NULL
XtNyRefWidget	XtCYRefWidget	Widget	the parent form
XtNyOffset	XtCYOffset	int	0
XtNyAddHeight	XtCYAddHeight	Boolean	FALSE
XtNyVaryOffset	XtCYVaryOffset	Boolean	FALSE
XtNyResizable	XtCYResizable	Boolean	FALSE
XtNyAttachBottom	XtCYAttachBottom	Boolean	False
XtNyAttachOffset	XtCYAttachOffset	int	0

XtNxRefName XtNyRefName

When a widget is added as a child of the form its position is determined by the widget it references. The reference widget must be created before the widget which references it is created. These resources allow the name of the reference widget to be given. The form converts this name to a widget to use for the referencing. Any widget that is a direct child of the form or the form widget itself can be used as a reference widget.

XtNxRefWidget XtNyRefWidget

The application can specify the reference widget as either a string representing the name of the widget (as described above) or as the Widget ID value returned from XtCreateWidget. This resource is the means by which a widget ID is specified.

XtNxOffset XtNyOffset

The location of a widget is determined by the widget it references. As the default, a widget's position on the form exactly matches its reference widget's location. There are two additional pieces of data used to determine the location. This resource defines an integer value representing the number of pixels to add to the reference widget's location when calculating the widget's

location.

XtNxAddWidth XtNyAddHeight
This resource indicates whether or not to add the width or height of the reference widget to a widget's location when determining the widget's position.

XtNxVaryOffset XtNyVaryOffset
When a form is resized, it processes the constraints contained within its children. This resource allows the spacing between a widget and the widget it references to vary (either increase or decrease) when a form's size changes. For widgets that directly reference the form widget this resource is ignored. The spacing between a widget and its reference widget can decrease to 0 pixels if the XtNAddWidth resource is FALSE or to 1 pixel if XtNAddWidth is TRUE.

XtNxResizable XtNyResizable
This resource specifies whether the form can resize (shrink) a widget. When a form's size becomes smaller the form will resize its children only after all of the inter-widget spacing of widget's with their VaryOffset resource set to TRUE. The form keeps track of a widgets initial size or size generated through XtSetValues so that when the form then becomes larger the widget will grow to it original size and no larger.

XtNxAttachRight XtNyAttachBottom
Widgets are normally referenced from "form left" to "form right" or from "form top" to "form bottom." The attach resources allow this reference to occur on the opposite edge of the form. These resources, when used in conjunction with the varyOffset resources, allow a widget to float along the right or bottom edge of the form. This is done by setting both the Attach and VaryOffset resources to TRUE. A widget can also span the width and height of the form by setting the Attach resource to TRUE and the VaryOffset resource to FALSE.

XtNxAttachOffset XtNyAttachOffset
When a widget is attached to the right or bottom edge of the form (through the above resources), the separation between the widget and the form is defaulted to 0 pixels. This resource allows that separation to be set to some other value. Also, for widgets that are not attached to the right or bottom edge of the form, this constraint specifies the minimum spacing between the widget and the form.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

EXAMPLES

The following examples list the desired layout of widgets within a form and the constraints needed to achieve the layout.

TitleBar

Use the following constraints to get a titlebar widget to span the top of a form the following constraints can be used. For a widget named *title* the .Xdefaults file will contain.

*title.xRefName:	"form widget name"	attach to the left edge of the form
*title.xOffset:	5	offset 5 pixels from the left edge
*title.xResizable:	TRUE	title is horizontally resizable
*title.xAttachRight:	TRUE	attach to the right edge of the form
*title.xAttachOffset:	5	offset 5 pixels from right edge
*title.yRefName:	"form widget name"	attach to the top edge of the form

Dynamic Scrolled Window

The above constraints work generally for any widget type that is to span the form and that need to be resized as the form increases or decreases in size. For example, if the child widget is a scrolled window named *sWin* that dynamically resizes as the form resizes in both the horizontal and vertical directions the constraints are as follows.

*sWin.xRefName:	"form widget name"	attach to the left edge of the form
*sWin.xOffset:	5	offset 5 pixels from the left edge
*sWin.xResizable:	TRUE	scrollWin is horizontally resizable
*sWin.xAttachRight:	TRUE	attach to the right edge of the form
*sWin.xAttachOffset:	5	offset 5 pixels from right edge
*sWin.yRefName:	"form widget name"	attach to the top edge of the form
*sWin.yOffset:	5	offset 5 pixels from the left edge
*sWin.yResizable:	TRUE	scrollWin is vertically resizable
*sWin.yAttachRight:	TRUE	attach to the bottom edge of the form
*sWin.yAttachOffset:	5	offset 5 pixels from right edge

Right or Bottom Attached Widgets

For a widget named *widget* to float along the right or bottom edge of the form as it is resized the constraint set is the same as for the titlebar example with the following changes.

*widget.xRefName:	"any widget name"	the widget to the left of this one
*widget.varyOffset:	TRUE	adjust the spacing with the reference widget

```
*w0,0.xRefName:     "form widget name"
*w0,0.xOffset:      5
*w0,0.xResizable:   TRUE
*w0,0.yRefName:     "form widget name"
*w0,0.yOffset:      5
*w0,0.yResizable:   TRUE

*w0,1.xRefName:     widget0,0
*w0,1.xResizable:   TRUE
*w0,1.yRefName:     widget0,0
*w0,1.yOffset:      5
*w0,1.yAddHeight:   TRUE
*w0,1.yResizable:   TRUE

*w1,0.xRefName:     widget0,0
*w1,0.xOffset:      20
*w1,0.yAddWidth:    TRUE
*w1,0.xResizable:   TRUE
*w1,0.yRefName:     widget0,0
*w1,0.yOffset:      5
*w1,0.yAddHeight:   TRUE
*w1,0.yResizable:   TRUE

*w1,1.xRefName:     widget1,0
*w1,1.xResizable:   TRUE
*w1,1.yRefName:     widget1,0
*w1,1.yOffset:      5
*w1,1.yAddHeight:   TRUE
*w1,1.yResizable:   TRUE
```

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), COMPOSITE(3X), CONSTRAINT(3X), XWMANAGERCLASS(3X)

NAME

XwframeWidgetClass – the X Widget's frame widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Frame.h>
```

CLASSES

The Frame widget is built from the Core, Composite, and XwManager classes.

The widget class to use when creating a frame is **XwframeWidgetClass.**

The class name for frame is **Frame.**

DESCRIPTION

The Frame widget is a very simple manager used to enclose a single child in a border drawn by the Frame widget. It uses the XwManager class resources for border drawing and performs geometry management such that its size will always match its child size plus the highlightThickness defined for it.

Frame is most often used to enclose other managers when the application developer desires the manager to have the same border appearance as the primitive widgets. Frame can also be used to enclose primitive widgets that do not support the same type of border drawing. This will give visual consistency when developing applications using diverse widget sets.

NEW RESOURCES

The Frame widget does not define any resources.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X)

NAME

XwimageEditWidgetClass – the X Widget's image editor widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/ImageEdit.h>
```

CLASSES

ImageEdit is built from the Core and Primitive classes.

The widget class to use when creating an image editor is **XwimageEditWidgetClass.**

The class name is **ImageEdit.**

DESCRIPTION

The ImageEdit widget allows an image to be displayed in an enlarged format so that it may be edited on a pixel-by-pixel basis. The specified image is displayed in a grid structure so that a user may see and modify the composition.

To change the image, the user moves the mouse to the desired point and presses the mouse button. The pixel under the cursor will change to the foreground color. If the cursor is moved while the button is pressed, all pixels that are touched will change to the foreground color.

NEW RESOURCES

The ImageEdit defines a set of resource types that can be used by the programmer to control the appearance and behavior of the widget. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string name. The following table contains the set of resources defined by ImageEdit.

ImageEdit Resource Set			
Name	Class	Type	Default
XtNimage	XtCImage	XImage *	NULL
XtNpixelScale	XtCPixelScale	int	6
XtNgridThickness	XtCGridThickness	int	1
XtNdrawColor	XtCBackground	Pixel	Black
XtNeraseColor	XtCBackground	Pixel	White
XtNeraseOn	XtCEraseOn	Boolean	True
XtNbackground	XtCBackground	Pixel	Black

XtNimage
This is a pointer to the image that is displayed in the grid. It points to an XImage structure.

XtNpixelScale
This resource defines the magnification factor to use when displaying the expanded image.

XtNgridThickness
This resource defines the separation between the magnified pixels.

XtNdrawColor
This resource define the color to be used for drawing in the widget.

XtNeraseColor
This resource defines the color used for erasing in the widget. Erase is enabled by the **eraseOn** resource. When selections occur on the widget, the widget determines the color of the pixel selected. If the selected pixel is not the same as the draw color, the draw color will be used to draw until the button release occurs. If the selected pixel is the draw color, the erase color will be used for drawing until the button release occurs.

XtNeraseOn
This resource is a boolean variable that indicates whether erasing is enabled or not. If set to TRUE, drawing will occur as described above. If set to FALSE, only the draw color will be used for drawing.

XtNbackground
ImageEdit redefines the core class background resource to default it to the color black. This is then used as the background color for the widget's window which will be reflected in the grid color.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

The following translations are defined for the ImageEdit widget.

```
<BtnDown>:          select()
<BtnUp>:            release()
Button1<PtrMoved>:  moved()
<EnterWindow>:      enter()
<LeaveWindow>:      leave()
```

ACTIONS

select: Selections occurring on an image edit cause drawing or erasing on the selected pixel, activate the moved action for continuous drawing and invoke the primitive class XtNselect callback functions.

release: Release concludes a drawing sequence and invokes primitive class XtNrelease callbacks.

moved: Moved causes drawing or erasing to occur from the last cursor position to the current cursor position.

enter: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the image edit's border will be highlighted. Otherwise no action is taken.

leave: If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the image edit's border will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X)

NAME

XwlistWidgetClass – the X Widget's list manager widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/SWindow.h>
#include <Xw/List.h>
```

CLASSES

List is built from the Core, Composite, Constraint, XwManager and XwSWindow classes.

The widget class to use when creating a list manager is **XwlistWidgetClass.** The class name is **List.**

DESCRIPTION

The List widget allows a two-dimensional set of widgets to be presented to the user in a rows/columns fashion. The layout will typically consist of n columns, not all of which need to be visible on the screen at one time. Each column will have some number of objects, such as labels or icons, arranged vertically. Separate columns may have unequal numbers of members--column A may have 10 elements, while column B has 17 elements. All members of each column are not required to be visible on the screen. The entire list window can be scrolled either vertically or horizontally, but the individual columns cannot be individually scrolled. If an application needs to have columnar scrolling, it may instanstiate multiple List widgets, each having only one column.

By default, each column is wide enough to display the longest item in the data. A resource is availible to allow each column to be a fixed width, with the excess characters being clipped. When the List widget is shrunk by a Resize call, columns that are beyond the right edge of the new size will be clipped. List elements are also adjusted to force a common height, with each element being set to the height of the tallest member of the column. This automatic sizing can be turned off through a resource, or forced to an arbitrary height. If a constant height is selected, any element that will not fit in the specified space will be clipped.

The List widget provides management and layout functions for its elements, as well as a means for the user to choose elements, and allows an application to be notified when those elements are selected. However, it is the responsibility of the application to create the actual widgets that are to be inserted into the list. The widgets may be of any type, but currently only Primitive class widgets will work correctly.

To construct a list, the application must create each element as a child of the List widget. The row and column position of the element can be specified by means of a constraint resource. If the row and column are not given, the list will be constructed as a one column by n row structure. The List widget will fill in the position of the element and store it in the constraint record so that it may be examined later.

The List widget supports two methods of choosing an item from its displayed list: single and multiple. A resource controls which mode is currently active.

In single choice mode, the user may move the cursor onto any element in the list and click the mouse button defined as "Begin Select." By default, this is the left button. When the button is pressed, the list item is highlighted. If the user drags the mouse with the button held down, the highlighted selection will track the pointer. If the pointer moves off the currently highlighted item, it will become unselected,

returning to its original state, and the item that the pointer has moved onto becomes highlighted. When the user releases the button, the currently selected item becomes the "choice," and the List widget invokes the select callback associated with the chosen item. The application must take over the widget's select callback in order to be notified that the item has been selected.

Multiple item selection is designed to allow the user to easily select several elements from the displayed list. When the user presses the mouse button bound to "Begin Select," the item currently under the pointer is highlighted to indicate that it is included in the selection set. As the user drags the mouse with the button down, the original choice remains highlighted, and any new items that the pointer touches also becomes highlighted. At any time, the user can "back up" the selection by leaving an item on the same side as it was entered. When the user finally releases the button, all highlighted elements are marked as chosen, and the selection callback is invoked for each item.

Selections can be either "sticky" or "instant." The selection mode is set through a resource. If set to sticky, the selection will remain highlighted after the user releases the mouse button, and will not be cleared until the next button press. In instant mode, the highlight will dissappear when the button is released.

The selection mechanism can be affected by a "bias" that is controlled through a widget resource. The allowable bias types are row, column, and none (default). In this mode only list items that are actually touched with the pointer are included in the selection. In Row Bias mode. entire rows of items may be selected by moving the pointer vertically within a column. For example, consider the following case:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

If the user pressed the mouse button when the pointer was over item 3 and then moved to item 9, items 4 thru 8 would also be highlighted. In Column Bias mode, entire columns can be selected by movement between rows. Using the above diagram, and assuming column bias, if the user clicked on item 2 and moved to item 3, elements 8 and 14 would also be selected.

Additional selections can be made without disturbing the original by following the above procedures, by depressing the button bound to the "Begin Extended Choice" function (which is defined as SHIFT + Left button in the default case).

The visual effect of highlighting can be accomplished in two ways: simple border highlighting, and inverse video. This may be configured through a List widget resource. Both styles are necessary--the inverse style of highlighting is by far the most common and natural interface, but could possibly conflict with an application or window manager that uses inverse to indicate the X11 "selection." The default highlighting style is inverse.

A user can select items that are not currently visible by simply extending the selection out of the visible window in the desired direction. The list will automatically scroll under the selection as needed, until there are no more list elements availible in the given direction. For example, in single-selection mode, if the user were to begin the selection on a visible element, and then drag the cursor down the column past the last visible item, the window would scroll up to display further choices.

When a list element is destroyed, the list will be re-ordered according to the value of the XtNdestroyMode resource. When it is XwSHRINK_COLUMN (the default), all list elements below the affected widget and in the same column will be moved up one row, and their row constraint resources will be updated to reflect the new positioning. When this resource is set to XwSHRINK_ALL, the elements will be moved in a row-wise fashion to fill the spot left by the affected element. The widget to the right of the affected one will be moved to the left, and so on to the last column. The first element of the next row will be moved into the last spot on the current column. This process will continue for all remainin g rows in the list. If the value of this resource is XwNO_SHRINK, the list will not change its ordering and a "hole " will appear in the place of the affected element.

NEW RESOURCES

The List widget defines a unique set of resource types which can be used by the programmer to control the appearence and behavior of the list. The programmer can also set the values for the Core, Composite, Constraint, Manager and SWindow widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, drop the XtN from the resource name. The following table contains the set of resources defined by List.

List Resource Set			
Name	**Class**	**Type**	**Default**
XtNnumColumns	XtCNumColumns	int	1
XtNcolumnWidth	XtCColumnWidth	int	0
XtNelementHeight	XtCElementHeight	int	0
XtNselectionStyle	XtCSelectionStyle	int	XwINSTANT
XtNselectionMethod	XtCSelectionMethod	int	XwSINGLE
XtNselectionBias	XtCSelectionBias	int	XwNO_BIAS
XtNelementHighlight	XtCElementHighlight	int	XwBORDER
XtNdestroyMode	XtCDestroyMode	int	XwSHRINK_COLUMN
XtNnumSelectedElements	XtCNumSelectedElements	int	0
XtNselectedElements	XtCSelectedElements	WidgetList*	NULL

XtNnumColumns
The number of columns in the list.

XtNcolumnWidth
The width of each column. If the value is 0, the width defaults to the width of the largest element.

XtNelementHeight
The height of each element. Zero implies that each element is resized to the height of the tallest element.

XtNselectionStyle
Controls the type of selection - either XwINSTANT or XwSTICKY.

XtNselectionMethod
Controls the selection mode - either one element at a time (XwSINGLE) or multiple (XwMULTIPLE).

XtNselectionBias
Bias mode - either XwNO_BIAS, XwROW_BIAS or XwCOL_BIAS.

XtNelementHighlight
This controls the highlight mode on selection - either border highlighting (XwBORDER) or inversion (XwINVERT).

XtNdestroyMode
Controls the visual appearance of the list when an element is deleted. One of XwSHRINK_COLUMN, XwSHRINK_ALL or XwNO_SHRINK.

XtNSelectedElements
This is a list of the widgets currently marked as selected. An application program can issue a call to XtGetValues on this resource at any time to query the selected elements.

XtNnumSelectedElements
The number of widgets currently selected (in the list pointed to by XtNselectedElements).

CONSTRAINT RESOURCES

The following resources are attached to every widget inserted into List. Refer to CONSTRAINT(3X) for a general discussion of constraint resources.

Constraint Resource Set -- Children of XWLIST(3X)			
Name	Class	Type	Default
XtNrowPosition	XtCRowPosition	int	-1
XtNcolumnPosition	XtCColumnPosition	int	-1

XtNrowPosition,XtNcolumnPosition
This is the row,column location of the element in the list. If these values are greater than or equal to zero, the widget is inserted into the list at that position. If the values are left at -1, the List widget will create a list with XtNnumColumns number of columns, assigning row and column positions as needed.

INCORPORATED RESOURCES

No incorporated resources are currently exported by the List widget.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

ScrolledWindow Resource Set - XWSCROLLEDWINDOW(3X)			
Name	Class	Type	Default
XtNvsbX	XtCVsbX	int	-1
XtNvsbY	XtCVsbY	int	-1
XtNvsbWidth	XtCVsbWidth	int	20
XtNvsbHeight	XtCVsbHeight	int	285
XtNhsbX	XtCHsbX	int	-1
XtNhsbY	XtCHsbY	int	-1
XtNhsbWidth	XtCHsbWidth	int	285
XtNhsbHeight	XtCHsbHeight	int	20
XtNforceHorizontalSB	XtCForceHorizontalSB	Boolean	FALSE
XtNforceVerticalSB	XtCForceVerticalSB	Boolean	FALSE
XtNvScrollEvent	XtCCallBack	Pointer	NULL
XtNhScrollEvent	XtCCallBack	Pointer	NULL
XtNinitialX	XtCInitialX	int	0
XtNinitialY	XtCInitialY	int	0

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the list widget.

TRANSLATIONS

The translations used for List are as follows:

```
<EnterWindow>:      enter()
<LeaveWindow>:      leave()
```

ACTIONS

enter: Enter window events occurring on the list window are handled by this action.

leave: Leave window events occurring on the list window are handled by this action.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), COMPOSITE(3X), CONSTRAINT(3X), XWMANAGERCLASS(3X), XWSCROLLEDWINDOW(3X)

NAME

XwmanagerWidgetClass – X Widget Manager MetaClass

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
```

CLASSES

The XwManagerClass is built from the Core, Composite and Constraint classes.

DESCRIPTION

The manager class is an X Widget meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other widget classes. It provides methods (procedures) which handle keyboard traversal and border highlighting for other manager widgets.

NEW RESOURCES

The manager class defines a set of resources used by the programmer to specify data for widgets which are subclasses of Manager. The string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off the resource name. For instance, XtNtraversalOn becomes traversalOn.

Manager Resource Set			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

XtNforeground
This resource defines the foreground color for the widget. Widgets built upon this class can use the foreground for their drawing.

XtNbackgroundTile
This resource defines the tile to be used for the background of the widget. It defines a particular tile to be combined with the foreground and background pixel colors. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNhighlightThickness
This resource specifies an amount of border spacing around the border of the widget. It is typically used by managers to have padding space around their children and to draw special borders. This highlight thickness is and an integer value representing the width, in pixels, of the border area. This value must be greater than or equal to 0.

XtNtraversalOn
The application can define whether keyboard traversal is active or not. The default for this resource is typically FALSE.

XtNlayout
This flag controls how the manager widget's geometry deals with too little or too much space. The valid settings for this field are XwMINIMIZE, XwMAXIMIZE and XwIGNORE. (The counterpart for these settings to be used in resource files, like .Xdefaults, are: minimize, maximize and ignore.) Typically, the XwMINIMIZE means to request the minimum amount of space necessary to display all children. The XwMAXIMIZE means that if additional space is given to the widget (i.e., at create time or set values time) then use the additional space as padding between children widgets. The XwIGNORE settings means, maintain the size set at create time or at set value time and never change size in response to a child widget's request (i.e., added/deleted/modified a child widget). Look at the description of the individual manager widgets to see if this feature is supported.

XtNnextTop
This callback procedure is used by the applications programmer to move the focus from one toplevel widget to another toplevel widget.

NOTE: The XwManagerClass provides a specialized insert child procedure. Manager widgets for which ordering makes sense (such as the RowCol manager widget) make use of the procedure. It allows an application to provide a special argument list type XtNinsertPosition with an integer value. This value specifies where in the child list the new widget is inserted.

KEYBOARD TRAVERSAL

If the traversalOn resource is TRUE (either when the widget is created or during a call to XtSetValues) the manager widget's translation table is augmented with the following translations:

<EnterWindow>:	enter()
<LeaveWindow>:	leave()
<Visible>:	visible()
<FocusIn>:	focusIn()

ACTIONS

enter:
If the widget is a top level manager and traversal is on, then begin or resume traversal.

leave:
If the widget is a top level manager and traversal is on, then suspend traversal.

visible:
If traversal is on for a widget of this class and the widget that is focused becomes hidden (e.g. another window obscures its visibility), then the focus moves to another visible widget.

focusIn:
If the widget is a top level manager and traversal is on, then begin traversal.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X)

NAME

XwmenubuttonWidgetClass – the X Widgets menubutton widget.

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/MenuBtn.h>

CLASSES

The menubutton widget is built out of the Core, XwPrimitive, and XwButton classes.

The widget class to use when creating a menupane is **XwmenubuttonWidgetClass.** The class name is **MenuButton.**

DESCRIPTION

The menubutton widget is commonly used with menupane and menu manager widgets to build a menu system. The menubutton consists of a single label, a mark and a cascade indicator. The menubutton is broken into three areas. Starting from the left border of the menubutton the areas are: the mark area, the label area and the cascade area. By default, the mark area contains a checkmark image, the label area contains the name of the menubutton widget and the cascade area contains an arrow image. The label can be set to any string or image and the label area attempts to grow or shrink to accommodate it. The mark and cascade can be set to an image, although the width of the these areas remains fixed.

The default semantic for this button is that button 1 down causes the select call backs to be invoked. When a menubutton is used in a menu manager, this may be overridden by the menu manager. The select callbacks may also be invoked by a keyboard accelerator or mnemonic, although, it is up to the menu manager to determine whether the accelerator or mnemonic is active.

The menubutton is often used with a menupane and menu manager widget although it is not necessary. The menubutton could simply be used as another button widget.

NEW RESOURCES

The MenuButton widget defines a set of resource types used by the programmer to specify the data for the menubutton. The programmer can also set the values for the Core, Primitive and Button widget classes to set attributes for this widget. The following table contains the set of resources defined by MenuButton. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name.

MenuButton Resource Set			
Name	**Class**	**Type**	**Default**
XtNlabelType	XtCLabelType	int	XwSTRING
XtNlabelImage	XtCLabelImage	XImage *	NULL
XtNcascadeImage	XtCCascadeImage	XImage *	NULL
XtNcascadeOn	XtCCascadeOn	Widget	NULL
XtNmarkImage	XtCMarkImage	XImage *	NULL
XtNsetMark	XtCSetMark	Boolean	FALSE
XtNkbdAccelerator	XtCKbdAccelerator	String	NULL
XtNmnemonic	XtCMnemonic	String	NULL
XtNmgrOverrideMnemonic	XtCMgrOverrideMnemonic	Boolean	TRUE
XtNmenuMgrId	XtCMenuMgrId	Widget	NULL
XtNcascadeSelect	XtCCallback	Pointer	NULL
XtNcascadeUnselect	XtCCallback	Pointer	NULL

XtNlabelType
Two styles of labels are supported by the MenuButton widget: text string labels and image labels. The text string label is defined by the Button resource XtNlabel and the image label is defined by the XtNlabelImage resource. To programmatically set this resource, use either the XwSTRING define or the XwIMAGE define. To set this resource using the .Xdefaults files, use one of the strings string or image.

XtNlabelImage
If XtNlabelType indicates that a label image should be displayed, then this resource contains the image used. This is a pointer to an XImage structure which describes the label image data. If the image is defined with XYBitmap data, then the image is nicely inverted when the menubutton is highlighted.

XtNcascadeImage
This resource points to an XImage structure which describes the cascade image data. The cascade area is a fixed size (16x16). If this resource is set to NULL, then the default cascade image, an arrow, is used. The cascade indicator is not displayed if the XtNcascadeOn resource is set to NULL. If the image is defined with XYBitmap data, then the image is nicely inverted when the menubutton is highlighted.

XtNcascadeOn
This resource determines if the cascade indicator is displayed. It is typically set only by the menu manager and contains the widget ID of the menupane which cascades as a submenu from this menubutton. This resource is set to NULL to disable the display of the cascade indicator.

XtNmarkImage
This resource points to an XImage structure which describes the mark image data. The mark area is a fixed size (16x16). If this resource is set to NULL, then the default mark image is used. The mark is not displayed if the XtNsetMark resource is set to FALSE. If the image is defined with XYBitmap data, then the image is nicely inverted when the menubutton is highlighted.

XtNsetMark
This boolean resource determines whether the mark is displayed.

XtNkbdAccelerator
This resource is a string which describe a set of modifiers and the key which may be used to select this menubutton widget. The format for this string is identical to that used by the translations manager, with the exception that only a single event may be specified, and only KeyPress events are allowed. If the menubutton does not have a menu manager associated with it, then this resource is ignored. The menu manager determines when, and if, this accelerator is available.

XtNmnemonic
Certain menu managers allow the menubuttons to have a mnemonic. Mnemonics provide the user with another means for selecting a menu button. This resource is a NULL terminated string, containing a single character. The menu manager determines if this mnemonic is available. If the XtNmgrOverrideMnemonic resource is false and the mnemonic is found in the label string, then that character is underlined when the menubutton is displayed. Refer to XwPullDown(3X) man page for further discussion of traversal.

XtNmgrOverrideMnemonic
This boolean resource determines if the mnemonic character is underlined in the label string. If it is set to TRUE, then the mnemonic character is not underlined. This resource is typically set only by menu managers.

XtNmenuMgrId
This resource is used only by menu managers to indicate to the menubutton widget its menu manager. If this is set to NULL, then the menubutton checks if it has a menu manager at the appropriate level in its parentage. This resource should not be set by users.

XtNcascadeSelect
This resource provides the means for registering callback routines which are invoked if a cascade indicator is displayed and the pointer moves into the cascade area. In some cases, the menu manager suppresses the calling of these callback routines. The menubutton does not pass any data in the call_data field of the callback.

XtNcascadeUnselect
This resource provides the means for registering callback routines which are invoked if a cascade indicator is displayed and the pointer moves out of the cascade area. These callbacks are only invoked if the XtNcascadeSelect callbacks have been previously invoked. The menubutton passes data in the call_data field of the callback. It is a pointer to the XwunselectParams data structure shown below:

```
typedef struct
{
  Dimension             rootX;
  Dimension             rootY;
  Boolean               remainHighlighted;
} XwunselectParams;
```

The rootX and rootY parameters have the position of the pointer relative to the root window when the event occurred which caused the XtNcascadeUnselect call backs to be called. The remainHighlighted parameter is used by cascading submenus. It is set by the menu manager's call back routine to indicate that the pointer traversed from a cascade into the submenu. If the boolean is set TRUE, then the menubutton does not unhighlight on

exit. It also sets up an event handler on its parent menupane so that it is notified if the pointer enters another menubutton, in which case the menubutton should then unhighlight.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

Button Resource Set -- XWBUTTON(3X)			
Name	Class	Type	Default
XtNfont	XtCFont	XFontStruct *	Fixed
XtNlabel	XtCLabel	caddr_t	widget name
XtNlabelLocation	XtCLabelLocation	int	XwRIGHT
XtNvSpace	XtCVSpace	int	2
XtNhSpace	XtCHSpace	int	2
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground

TRANSLATIONS

The default translations set by the menubutton widget are as follows:

<Btn1Down>:	select()
<EnterWindow>	enter()
<LeaveWindow>:	leave()
<Motion>:	moved()
<Key>Select:	select()
<Key>Up:	traverseUp()
<Key>Down:	traverseDown()
<Key>Left:	traverseLeft()
<Key>Right:	traverseRight()
<Key>Next:	traverseNext()
<Key>Prior:	traversePrev()
<Key>Home:	traverseHome()
<Visible>:	visibility()
<Unmap>:	unmap()
<Key>KP_Enter:	traverseNexttop()

ACTIONS

select:

If a menu manager is present, then it is informed of the select event. The menu manager indicates whether this select event should be processed or ignored. If there is no menu manager, or if the menu manager indicates the event is to be processed, then the select callbacks are called.

enter:

If a menu manager is present, then it is informed of the enter event. The menu manager indicates whether this enter event should be processed or ignored. If there is no menu manager present, or if the menu manager indicates the event is to be processed, then the menubutton is highlightedi A processed enter action also calls the moved action to determine if the pointer is in the cascade indicator area.

leave:

If a menu manager is present, then it is informed of the leave event. The menu manager indicates whether this leave event should be processed or ignored. If there is no menu manager present, or if the menu manager indicates that the leave event is to be processed, then the menubutton is unhighlighted. If the XtNcascadeSelect callbacks have been called, the XtNcascadeUnselect callbacks are called.

moved:

If this menubutton has cascading on, then this action determines if the pointer is in the cascade area and calls the XtNcascadeSelect or XtNcascadeUnselect callbacks if necessary.

traverseUp:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the menu button positioned above the current traversal item; wrap to the bottom, if necessary.

traverseDown:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the menu button positioned below the current traversal item; wrap to the top, if necessary.

traverseLeft:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the menupane cascading from this menubutton, if one is present.

traverseRight:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the menupane from which the current one has cascaded.

traverseNext:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the next menu tree, if one is present.

traversePrev:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the previous menu tree, if one is present.

traverseHome:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the first menupane in the menu hierarchy.

visibility:
This action routine overrides the visibility action routine provided by the XwPrimitive meta class.

unmap:
This action overrides the unmap action routine provided by the XwPrimitive meta class.

traverseNexttop:
Inform the menu manager controlling this widget that it should transfer the keyboard focus to the next top level menupane.

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3), XWPRIMITIVE(3X), XWBUTTON(3X)

NAME

XwmenumgrWidgetClass – the X Widgets menu manager meta widget.

SYNOPSIS

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>

CLASSES

The menu manager class is built the Core, Composite, Constraint and XwManager classes.

DESCRIPTION

The MenuMgr class is an X Widget meta class. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other menu manager widget classes.

NEW RESOURCES

The menu manager defines a set of resource types which may be used by the programmer to specify the data for widgets which are a subclass of MenuMgr. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by MenuMgr.

MenuMgr Resource Set			
Name	Class	Type	Default
XtNassociateChildren	XtCAssociateChildren	Boolean	TRUE
XtNmenuPost	XtCMenuPost	String	"<Btn1Down>"
XtNmenuSelect	XtCMenuSelect	String	"<Btn1Up>"
XtNmenuUnpost	XtCMenuUnpost	String	NULL
XtNkbdSelect	XtCKBDSelect	String	"<Key>Select"

XtNassociateChildren
This resource indicates whether the menu hierarchy controlled by the menu manager is accessible only from within the associated widget, or from within the widget and any of the widget's children.

XtNmenuPost
This string resource describes the button event and any required modifiers needed to post one of the top level menupanes controlled by the menu manager. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a ButtonPress or ButtonRelease event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event occurs.

XtNmenuSelect
This string resource describes the button event and any required modifiers needed to select a menu button within any of the menupanes controlled by the menu manager. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a ButtonPress or ButtonRelease event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event

occurs.

XtNmenuUnpost
This string resource describes the key event and any required modifiers needed to unpost the currently viewable set of menupanes controlled by the menu manager. This provides the user with the means for unposting a menu hierarchy from the keyboard, without selecting a menu button. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a key event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event occurs.

XtNkbdSelect
This string resource describes the key event and any required modifiers needed to select the currently highlighted menu button. This provides the user with the means for selecting a menu item from the keyboard, without being required to use the mouse. The string is specified using the syntax supported by the Xt Intrinsic's translation manager, with three exceptions. First, only a single event may be specified. Secondly, the event must be a key event. Thirdly, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the button event occurs.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X)

NAME

XtMapWidget, XtSetMappedWhenManaged, XtUnmapWidget – map and unmap widgets

SYNTAX

XtMapWidget(*w*)
Widget *w*;

void XtSetMappedWhenManaged(*w*, *map_when_managed*)
Widget *w*;
Boolean *map_when_managed*;

XtUnmapWidget(*w*)
Widget *w*;

ARGUMENTS

map_when_managed
Specifies a Boolean value that indicates the new value of the map_when_managed field.

w Specifies the widget.

DESCRIPTION

If the widget is realized and managed and if the new value of map_when_managed is **True**, **XtSetMappedWhenManaged** maps the window. If the widget is realized and managed and if the new value of map_when_managed is **False**, it unmaps the window. **XtSetMappedWhenManaged** is a convenience function that is equivalent to (but slightly faster than) calling **XtSetValues** and setting the new value for the mappedWhenManaged resource. As an alternative to using **XtSetMappedWhenManaged** to control mapping, a client may set mapped_when_managed to **False** and use **XtMapWidget** and **XtUnmapWidget** explicitly.

SEE ALSO

XtManageChildren(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtMakeGeometryRequest, XtMakeResizeRequest – make geometry manager request

SYNTAX

XtGeometryResult XtMakeGeometryRequest(*w, request, reply_return*)
Widget *w;*
XtWidgetGeometry **request;*
XtWidgetGeometry **reply_return;*

XtGeometryResult XtMakeResizeRequest(*w, width, height, width_return, height_return*)
Widget *w;*
Dimension *width, height;*
Dimension **width_return, *height_return*

ARGUMENTS

reply_return	Returns the allowed widget size or may be NULL if the requesting widget is not interested in handling **XtGeometryAlmost**.
request	Specifies the desired widget geometry (size, position, border width, and stacking order).
w	Specifies the widget that is making the request.
width_return *height_return*	Return the allowed widget width and height.

DESCRIPTION

Depending on the condition, **XtMakeGeometryRequest** performs the following:

- If the widget is unmanaged or the widget's parent is not realized, it makes the changes and returns **XtGeometryYes**.
- If the parent is not a subclass of **compositeWidgetClass** or the parent's geometry_manager is NULL, it issues an error.
- If the widget's being_destroyed field is **True**, it returns **XtGeometryNo**.
- If the widget x, y, width, height and border_width fields are all equal to the requested values, it returns **XtGeometryYes**; otherwise, it calls the parent's geometry_manager procedure with the given parameters.
- If the parent's geometry manager returns **XtGeometryYes** and if **XtCWQueryOnly** is not set in the request_mode and if the widget is realized, **XtMakeGeometryRequest** calls the **XConfigureWindow** Xlib function to reconfigure the widget's window (set its size, location, and stacking order as appropriate).
- If the geometry manager returns **XtGeometryDone**, the change has been approved and actually has been done. In this case, **XtMakeGeometryRequest** does no configuring and returns **XtGeometryYes**. **XtMakeGeometryRequest** never returns **XtGeometryDone**.

Otherwise, **XtMakeGeometryRequest** returns the resulting value from the parent's geometry manager.

Children of primitive widgets are always unmanaged; thus, **XtMakeGeometryRequest** always returns **XtGeometryYes** when called by a child of a primitive widget.

The **XtMakeResizeRequest** function, a simple interface to **XtMakeGeometryRequest**, creates a **XtWidgetGeometry** structure and specifies that width and height should change. The geometry manager is free to modify any of the other window attributes (position or stacking order) to satisfy the resize request. If the return value is **XtGeometryAlmost**, width_return and height_return contain a compromise width and height. If these are acceptable, the widget should immediately make an

XtMakeResizeRequest and request that the compromise width and height be applied. If the widget is not interested in **XtGeometryAlmost** replies, it can pass NULL for width_return and height_return.

SEE ALSO

XtConfigureWidget(3Xt), XtQueryGeometery(3Xt)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NAME

XtManageChildren, XtManageChild, XtUnmanageChildren, XtUnmanageChild – manage and unmanage children

SYNTAX

typedef Widget *WidgetList;

void XtManageChildren(*children, num_children*)
    WidgetList *children;*
    Cardinal *num_children;*

void XtManageChild(*child*)
    Widget *child;*

void XtUnmanageChildren(*children, num_children*)
    WidgetList *children;*
    Cardinal *num_children;*

void XtUnmanageChild(*child*)
    Widget *child;*

ARGUMENTS

child	Specifies the child.
children	Specifies a list of child widgets.
num_children	Specifies the number of children.

DESCRIPTION

The **XtManageChildren** function performs the following:

- Issues an error if the children do not all have the same parent or if the parent is not a subclass of **compositeWidgetClass.**
- Returns immediately if the common parent is being destroyed; otherwise, for each unique child on the list, **XtManageChildren** ignores the child if it already is managed or is being destroyed and marks it if not.
- If the parent is realized and after all children have been marked, it makes some of the newly managed children viewable:
  - Calls the change_managed routine of the widgets' parent.
  - Calls **XtRealizeWidget** on each previously unmanaged child that is unrealized.
  - Maps each previously unmanaged child that has map_when_managed **True.**

Managing children is independent of the ordering of children and independent of creating and deleting children. The layout routine of the parent should consider children whose managed field is **True** and should ignore all other children. Note that some composite widgets, especially fixed boxes, call **XtManageChild** from their insert_child procedure.

If the parent widget is realized, its change_managed procedure is called to notify it that its set of managed children has changed. The parent can reposition and resize any of its children. It moves each child as needed by calling **XtMoveWidget,** which first updates the x and y fields and then calls **XMoveWindow** if the widget is realized.

The XtManageChild function constructs a **WidgetList** of length one and calls **XtManageChildren.**

NAME

XwMoveFocus – move the keyboard focus (and the pointer) to a new toplevel widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

void XwMoveFocus (w)
Widget w;
```

ARGUMENTS

w This is the ID of the widget to which the application wishes to move the focus. It should be the toplevel widget in a widget hierarchy and it should be a subclass of XwManager.

DESCRIPTION

XwMoveFocus is a very specialized function which can be used to move the keyboard and pointer focus to another toplevel widget hierarchy. It is useful when an application using keyboard traversal has multiple toplevel widget hierarchies and wishes to be able to move between these hierarchies without using the pointer device. Specifically, this function will warp the pointer to (1,1) in the specified widget and will also make a call to XSetInputFocus (this is necessary for use with window managers using an explicit listener mode).

ORIGIN

Hewlett-Packard Company.

SEE ALSO

NAME

XwpushButtonWidgetClass – the X Widgets pushbutton widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/PButton.h>
```

CLASSES

The pushbutton widget is built from the Core, XwPrimitive and XwButton classes.

The widget class to use when creating a pushbutton is **XwpushButtonWidgetClass**. The class name is **PushButton.**

DESCRIPTION

The pushbutton widget consists of a text label surrounded by a button border.

By default, button 1 down will invert the interior of the button: the background will be filled with the foreground color and the text will be written in the background color. Button 1 down also sets the button state to TRUE and issues any XtNselect callbacks that have been registered. Button 1 up will repaint the button in the normal state, set the button state to FALSE and issue any XtNrelease callbacks that have been registered.

As mentioned above, the XtNselect and XtNrelease callbacks can be attached to this widget. This widget can also be set to respond to Enter and Leave window events by highlighting and unhighlighting the button. This widget is also capable of handling keyboard traversal. See the translations below for the default traversal keycodes.

A final feature is that by setting the XtNtoggle resource to TRUE the pushbutton can be made to act like a toggle button.

NEW RESOURCES

The pushbutton widget class defines a set of resource types that can be used by the programmer to specify data for widgets of this class. Recall that the string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNfont becomes font.

PushButton Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNtoggle	XtCToggle	Boolean	FALSE

XtNtoggle
If set to TRUE makes the pushbutton act like a toggle button.

INHERITED RESOURCES

The following resources are inherited from the named superclasses. The defaults used for the pushbutton when being created are as follows:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

Button Resource Set -- XWBUTTON(3X)			
Name	**Class**	**Type**	**Default**
XtNfont	XtCFont	XFontStruct *	Fixed
XtNlabel	XtCLabel	caddr_t	NULL
XtNlabelLocation	XtCLabelLocation	int	right
XtNvSpace	XtCVSpace	int	2
XtNhSpace	XtCHSpace	int	2
XtNset	XtCSet	Boolean	FALSE
XtNsensitiveTile	XtCSensitiveTile	int	75_foreground

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHTLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive

man page for a complete description of these translations. See the TRANSLATIONS section in this man page for a description of the translations local to the pushbutton widget.

TRANSLATIONS

The input to the pushbutton is driven by the mouse buttons. The default translation set defining this button is as follows:

<Btn1Down>:	select()	
<Btn1Up>:	release()	
<EnterWindow>:	enter()	
<LeaveWindow>:	leave()	
<KeyDown>Select:	select()	HP "Select" key
<KeyUp>Select:	unselect()	HP "Select" key

ACTIONS

Note that this widget contains some actions which are not bound to any events by the default translations. The purpose of these additional actions are to allow advanced users to alter the button semantics to their liking.

toggle:
Toggle the set state of the button (make it TRUE if it was FALSE, FALSE if it was TRUE). Redraw the pushbutton to reflect the current button setting (if set, invert the button, otherwise draw normally). If the current state of the button is set (TRUE) issue the XtNselect callbacks, if not set (FALSE) issue the XtNrelease callbacks. No additional data beyond the widget id and the specified closure is sent with these callbacks.

select:
Select sets the state of the button to TRUE. It also redraws the pushbutton to reflect the current setting. It then issues any XtNselect callbacks which have been registered. No additional data beyond the widget id and the specified closure is sent with these callbacks.

unselect:
Release sets the state of the button to FALSE. It also redraws the pushbutton to reflect the current setting. It then issues any XtNrelease callbacks which have been registered. No additional data beyond the widget id and the specified closure is sent with these callbacks.

enter:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the button will be highlighted. Otherwise no action is taken.

leave:
If the XtNtraversalType resouces has been set to XwHIGHLIGHT_ENTER then the button will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWBUTTON(3X)

NAME

XwPanelWidgetClass – An X Widget for creating panels.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Panel.h>
```

CLASSES

A subclass of Core, Composite, Constraint and XwManagerClass.

The widget class to use when creating a Panel widget is **XwpanelWidgetClass.**

The class name of Panel is **Panel.**

DESCRIPTION

Panel provides a simple creation mechanism for the creation of application windows and associated menus. The panel widget is also appropriate for application sub-windows.

Panel will manage three types of children. Panel may have at most one child of each type. The types are titlebar, menu, and work space. Children are associated with these types via constraint resources (see below). Panel ignores all extra or unknown children.

Panel lays out its children such that the child of type titlebar is on the top, the child of type menu is below, and the child of type work space is on the bottom. Display of the titlebar child can be optionally inhibitted if the panel is under the control of a window manager which provides titlebars.

When Panel has its width changed by its parent, the menu, if displayed, is allowed to pick its own height, the title remains the same height and the work space is diminished or enlarged to fill the remaining available space. When Panel has its height reduced by its parent, space is taken from the work space until the work space is completely hidden. Further reductions in the height of Panel are shared between the title and the menu. When Panel has its height increased by its parent, if either the title or the menu are less than their optimum height, they are given the space until they reach their optimum height for the given width. If both the title and the menu are at their optimum height all space is given to the work space.

The initial width of Panel is the widest of all its children (padding is taken into account). The initial height of Panel is the sum of the heights of all its children and their padding.

When an application is running in a Panel with a titling window manager, there is a possibility of double titling. Unfortunately, the application writer cannot know at the time of development whether or not the user will have a titling window manager. Panel has two resources which together allow runtime decisions about titling. The first, XtNtopLevel, indicates whether the Panel is a canidate for double titling. The application must always set this variable appropriately. The second resource, XtNdisplayTitle, indicates whether or not the Panel should display a title.

NEW RESOURCES

To specify any of these resources within a resource defaults file, simply drop the XtN prefix from the resource name. Panel defines the following new resources:

Panel Resource Set			
Name	**Class**	**Type**	**Default**
XtNtopLevel	XtCTopLevel	Boolean	TRUE
XtNdisplayTitle	XtCDisplayTitle	Boolean	TRUE
XtNvSpace	XtCVSpace	int	0
XtNhSpace	XtCHSpace	int	0
XtNtitleToMenuPad	XtCTitleToMenuPad	int	0
XtNworkSpaceToSiblingPad	XtCWorkSpaceToSiblingPad	int	0

XtNtopLevel
Indicates whether not the panel is a candidate for management by a window manager. This should always be set by the application.

XtNdisplayTitle
Ignored if XtNtopLevel is FALSE.

Otherwise, if XtNdisplayTitle is TRUE, the titlebar child will be displayed. If XtNdisplayTitle is FALSE, the titlebar child will not be displayed.

This resource should be set by the user in the resource defaults file. If the user runs the application without a window manager or with a non-titling window manager, this resource should be set to TRUE. If the user runs with a titling window manager this resource should be set to FALSE.

XtNvSpace
Padding between the top of the Panel and the top child in pixels, and between the bottom of the Panel and the bottom child in pixels.

XtNhSpace
Padding between the sides of the Panel and the sides of the displayed children.

XtNtitleToMenuPad
If both a title and a menu child are being displayed, the padding between them in pixels.

XtNworkSpaceToSiblingPad
The padding between the work space child and the sibling above it. If there is no title nor menu being displayed this resource is ignored.

CONSTRAINT RESOURCES

The following resources will be attached to every widget inserted into Panel. Refer to CONSTRAINT(3X) for a general discussion of constraint resources.

Constraint Resource Set -- Children of PANEL(3X)			
Name	**Class**	**Type**	**Default**
XtNwidgetType	XtCWidgetType	XwWidgetType	XwWORK_SPACE
XtNcausesResize	XtCCausesResize	Boolean	FALSE

XWPANEL(3Xh)

XtNwidgetType
Indicates to Panel what type of child it is. The possible values are, XwWORK_SPACE, specified in a resource defaults file as "work space", XwTITLE, specified in a resource defaults file as "title", and XwPULLDOWN, specified in a resource defaults file as "pulldown".

XtNcausesResize
Controls whether changes in the child geometry can cause the Panel to make a geometry request of its parent. If TRUE for only one child, Panel will request changes whenever that child requests changes. If TRUE for multiple children, Panel will request changes whenever any of that set of children grow, and when all of that set of children have shrunk.

The behavior of this resource can be nullified by setting XwNLayout to XwIGNORE.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtClayout	int	minimize

TRANSLATIONS

The default translation set defining is as follows:

```
<EnterWindow>:    enter()
<LeaveWindow>:    leave()
```

ACTIONS

enter: If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE), initiate keyboard traversal.

leave: If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE), terminate keyboard traversal.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X)

NAME

XwpopupmgrWidgetClass – the X Widgets popup menu manager widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/PopupMgr.h>
```

CLASSES

The popup menu manager widget is built from the Core, Composite, Constraint, XwManager and XwMenuMgr classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating an instance of the popup menu manager is **XwpopupmgrWidgetClass**. The class name is **PopupMgr**.

DESCRIPTION

The popup menu manager widget is a composite widget which is used by an application to manage a collection of menupanes. Even though the popup menu manager is a composite widget, it should never have any normal widget children. Instead, all of its children are popup shell children; the child of each of the popup shell widgets is a menupane. In addition, the parent of the popup menu manager must be a popup shell widget, whose parent is the widget to which the menu tree is being associated.

The popup menu manager manages a collection of menupane widgets, which have been organized into a hierarchical tree structure; the root of the tree is the top level menupane. When the user requests that the menu be posted, by generating a post event within the widget (or possibly one of the widget's children), the top level menupane is posted.

Once the menu manager has posted the top level menupane, it will remain posted until the user generates a select action; at that point, the menupanes will be removed from the display, and the selected menu button will perform any required actions. If the select occurs outside of a menu button, or if the user issues the menu unpost event, then the menupanes are simply unposted.

The menu manager supports a mode by which the menu hierarchy may be associated only with the specified widget, or it may be associated with the widget and all of its children (both present and future children). If the menu is associated with the widget and its children, then a menu post event which occurs in either the widget or one of its children, will cause the menu to be posted.

The menu manager also supports a commonly used menuing feature, referred to as sticky menus. When operating in sticky menu mode, the menu manager will remember the last menu button selected by the user. The next time the user requests that the menu system be posted, all of the menupanes, up to the one containing the previously selected menu button, will be posted.

The popup menu manager provides a keyboard interface to the menus, through the use of keyboard accelerators, for posting the menu and for selecting a menubutton from within one of the menupanes. This manager does not support keyboard mnemonics. When traversal is enabled, the standard keyboard traversal keys are also operational. Using the mouse, while traversal is enabled, may produce confusing results for the user; thus, operating in this fashion is discouraged.

The popup menu manager provides the application writer with a global function which may be used to programmatically post a top level menupane at a particular position relative to a specified widget. The calling sequence and parameters are shown below:

```
XwPostPopup (menuMgr, relativeTo, x, y)
XwPopupMgrWidget menuMgr;
Widget relativeTo;
Position x,y;
```

XwPostPopup() posts the top level menupane associated with the specified menu manager at the requested (x,y) position, relative to the specified widget. If the relativeTo parameter is set to NULL, then the position is assumed to be relative to the root window.

NEW RESOURCES

The popup menu manager defines a set of resource types used by the programmer to specify the data for the menu manager. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by PopupMgr.

PopupMgr Resource Set			
Name	**Class**	**Type**	**Default**
XtNstickyMenus	XtCStickyMenus	Boolean	FALSE
XtNpostAccelerator	XtCPostAccelerator	String	NULL

XtNstickyMenus
This resource controls whether the menu manager operates in sticky menu mode.

XtNpostAccelerator
This resource indicates the keyboard event that can be used to post the top level menupane. The string is specified using the syntax supported by the translation manager, with three exceptions. First, only a single event may be specified. Second, the event must be a KeyPress or KeyRelease event. Third, all modifiers specified are interpreted as being exclusive; this means that only the specified modifiers can be present when the key event occurs.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE

Menu Manager Resource Set -- XWMENUMGR(3X)			
Name	Class	Type	Default
XtNassociateChildren	XtCAssociateChildren	Boolean	TRUE
XtNmenuPost	XtCMenuPost	String	"<Btn1Down>"
XtNmenuSelect	XtCMenuSelect	String	"<Btn1Up>"
XtNmenuUnpost	XtCMenuUnpost	String	NULL
XtNkbdSelect	XtCKbdSelect	String	"<Key>Select"

BUGS

Due to limitations within the Xt Intrinsics, keyboard accelerators for posting a menu pane or for selecting a menu item do not work if the widget to which the menu manager is attached has traversal enabled.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XwMENUMGR(3X)

NAME

XwprimitiveWidgetClass – the X Widget's primitive widget meta class

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
```

CLASSES

The Primitive widget class is built out of the Core class.

DESCRIPTION

The Primitive class is an X Widget MetaClass. It is never instantiated as a widget. Its sole purpose is as a supporting superclass for other widget classes. It handles border drawing and highlighting, traversal activation and deactivation and various callback lists needed by primitive widgets.

NEW RESOURCES

Primitive defines a set of resource types used by the programmer to specify the data for widgets which are subclasses of Primitive.

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

XtNforeground
This resource defines the foreground color for the widget. Widgets built upon this class can use the foreground for their drawing.

XtNbackgroundTile
This resource defines the tile to be used for the background of the widget. It defines a particular tile to be combined with the foreground and background pixel colors. The #defines for setting the tile value through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNhighlightColor
This resource defines the color to be used in the highlighting drawn by Primitive around the exterior of the widget.

XtNhighlightStyle
Two styles of border highlighting are supported by Primitive. They include drawing the highlighting with a pattern and widget specific border drawing. To set the highlight style through an arg list, use the #define XwPATTERN_BORDER. To set the highlight style through the .Xdefaults file, use the string pattern_border.

XWPRIMITIVE(3Xh)

For Widget Writers: The highlighting style of XwWIDGET_DEFINED is used exclusively by widgets with special highlighting requirements that need to override the normal highlighting types. To use, the widget inserts a highlight and unhighlight procedure into its primitive class and forces the highlightStyle field in the primitive instance to the define XwWIDGET_DEFINED. The primitive class will then make the appropriate calls to the highlight and dehighlight functions.

XtNhighlightTile
When the highlight style is XwPATTERN_BORDER, one of several tiles can be used for the drawing. The #defines for setting the values through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNhighlightThickness
The width of the highlight can be set using this resource. It is specified as an integer value representing the width, in pixels, of the highlight to be drawn. This value must be greater than or equal to 0. Note that highlighting takes place within the window created for a widget and is separate from the window border.

XtNtraversalType
Three modes of border highlighting activation are supported by Primitive. They are, no highlighting, highlight on the cursor entering the widgets window and highlight for keyboard traversal. The last mode is used by the keyboard traversal mechanism to indicate the widget that is to receive all input occurring within the widget hierarchy. To set the traversal type through an arg list, one of three defines can be used. They are XwHIGHLIGHT_OFF, XwHIGHLIGHT_ENTER and XwHIGHLIGHT_TRAVERSAL. The strings that can be used to set this resource through the .Xdefaults file are highlight_off, highlight_enter, and highlight_traversal.

XtNrecomputeSize
This boolean resource indicates to a primitive widget whether it should recalculate its size when an application makes a XtSetValues call to it. If set to TRUE, the widget will perform its normal size calculations will may cause its geometry to change. If set to FALSE, the widget will not recalculate its size.

XtNselect
This is a reserved callback list used by widget subclasses built upon Primitive to implement there callback lists.

XtNrelease
This is a reserved callback list used by widget subclasses built upon Primitive to implement there callback lists.

KEYBOARD TRAVERSAL

If the traversalType resource is set to highlight_traversal (either when the widget is created or during a call to XtSetValues) the Primitive widget's translation table is augmented with the following translations:

<FocusIn>:	focusIn()	
<FocusOut>:	focusOut()	
<Visible>:	visibility()	
<Unmap>:	unmap()	
<Key>Up:	traverseUp()	HP Up arrow key.
<Key>Down:	traverseDown()	HP Down arrow key.
<Key>Left:	traverseLeft()	HP Left arrow key.
<Key>Right:	traverseRight()	HP Right arrow key.
<Key>Next:	traverseNext()	HP "Next" key.
<Key>Prior:	traversePrev()	HP "Prev" key.
<Key>Home:	traverseHome()	HP Home arrow key.
<Key>KP_Enter:	traverseNextTop()	HP "Enter" key.

ACTIONS

focusIn:
If traversal is on for a widget of this class then accept the keyboard focus and visually indicate it by highlighting the widget.

focusOut:
If traversal is on for a widget of this class then indicate that the widget no longer has the focus by unhighlighting the widget.

visibility:
If traversal is on for a widget of this class and the widget that is focused becomes hidden (e.g. another window obscures its visibility), then the focus moves to another visible widget.

unmap:
If traversal is on for a widget of this class and the widget that is focused becomes unmapped, then the focus moves to another mapped widget.

traverseUp:
Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget above this one.

traverseDown:
Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget below this one.

traverseLeft:
Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget to the left of this one.

traverseRight:
Inform the parent of a widget of this class that it should transfer keyboard focus to the first widget to the right of this one.

traverseNext:
Inform the parent of a widget of this class that it should transfer keyboard focus to the next child in the parent's list of children.

traversePrev:
Inform the parent of a widget of this class that it should transfer keyboard focus to the previous child in the parent's list of children.

traverseHome:
Inform the parent of a widget of this class that it should transfer keyboard focus to the child which is closest to the upper left hand corner of the parent. If that child already has the keyboard focus, then ask the grandparent of the widget to give the keyboard focus to whichever of its children which is closest to the upper left hand corner.

traverseNextTop:
Find the topmost parent in a widget of this class hierarch which is a subclass of XwManager and tell it to issues any XtNnextTop callbacks that have been registered with it. The purpose of this callback is to allow applications to move the keyboard focus between top level widget hierarchies of the same application.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWCREATETILE(3X)

NAME

XwpulldownWidgetClass - the X Widgets pulldown menu manager widget.

SYNOPSIS

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xw/Xw.h>
#include <Xw/Pulldown.h>
```

CLASSES

The pulldown menu manager widget is built out of the Core, Composite, Constraint, XwManager and XwMenuMgr classes. Note that the Constraint fields are not used in this widget and so are not listed in the resource tables below. Also, since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating an instance of the pulldown menu manager is **XwpulldownWidgetClass**. The class name is **Pulldown.**

DESCRIPTION

The pulldown menu manager widget is a composite widget which is used by an application to manage a collection of menupanes. Even though the pulldown menu manager is a composite widget, it should never have any normal widget children. Instead, all of its children are popup shell children; the child of each of the popup shell widgets is a menupane. In addition, the parent of the pulldown menu manager must be a popup shell widget, whose parent is the widget to which the menu tree is being associated.

The pulldown menu manager manages a collection of menupane widgets, which have been organized into a hierarchical tree structure; the root of the tree is referred to as the top level menupane. The pulldown menu manager creates a pulldown widget as a child of the widget to which the menu tree is associated; as the menu tree is constructed, titlebuttons will be added to the pulldown widget, thus providing the user with a means for posting a particular portion of the menu tree. As menupanes are added to the menu tree, if cascading submenus are allowed, then only those menupanes which cascade off of the top level menupane will be folded up as a first level menupane with a new titlebutton within the pulldown widget. If cascading submenus are not allowed, then all cascading menupanes will be folded up into a first level menupane with a new titlebutton.

When the user requests that the menu be posted, by generating a post event within one of the titlebuttons, the menupane associated with the indicated titlebutton is posted. As soon as a select event or an unpost event is generated, the menupanes are unposted.

Once the menu manager has posted a first level menupane, it will remain posted until either the user generates a select action, the user generates an unpost action, or the user moves the cursor into a different titlebutton. If the select action occurs, then the menupanes will be removed from the display, and the appropriate menubutton will perform any required actions. If the select action occurs outside of a menubutton, or if the unpost action is generated, then the menupanes are simply unposted. If the cursor was moved into a different titlebutton, then the menupanes associated with the previous titlebutton will be unposted, and the first level menupane for the new titlebutton will be posted.

The menu manager supports a mode by which the menu hierarchy may be associated only with the specified widget, or it may be associated with the widget and all of its children (both present and future children). If the menu is associated with the

widget and its children, then a keyboard accelerator which occurs in either the widget or one of its children, will cause the appropriate action to occur.

The pulldown menu manager provides a keyboard interface to the menus, through the use of mnemonics and keyboard accelerators. A mnemonic may be used to post any of the first level menupanes; a posting mnemonic is issued by typing the appropriate mnemonic character in the presence of the modifiers specified by the postAccelerator resource. Keyboard accelerators are supported for selecting a menubutton from within any of the menupanes; accelerators are always active, even if the corresponding menubutton is not currently displayed. Keyboard mnemonics may also be used for selecting a menubutton; however, a menubutton's mnemonic is only active if the menupane in which it resides in is currently displayed. The pulldown menu manager only allows the first level pulldown menupanes to have keyboard mnemonics for posting.

NEW RESOURCES

The pulldown menu manager defines a set of resource types which may be used by the programmer to specify the data for the menu manager. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To specify any of these resources within the .Xdefaults file, simply drop the XtN prefix from the resource name. The following table contains the set of resources defined by Pulldown.

Pulldown Resource Set			
Name	Class	Type	Default
XtNallowCascades	XtCAllowCascades	Boolean	TRUE
XtNpostAccelerator	XtCPostAccelerator	String	"Meta"
XtNpulldownBarId	XtCPulldownBarId	Widget	NULL

XtNallowCascades

This resource is used to control whether any of the top level pulldown menupanes may have other menupanes cascading off of them. This resource must be set to the desired value when the menu manager widget is first created; it cannot be modified after the widget has been created.

XtNpostAccelerator

This resource is used to specify the keyboard modifiers which must be present when one of the post mnemonics is issued by the user. This resource must be set to the desired value when the menu manager widget is first created; it cannot be modified after the widget has been created.

XtNpulldownBarId

This resource is a read-only resource, and provides the application with the means for obtaining the widget Id for the frame widget which encloses the pulldown menubar widget. Applications should not use this to modify the attributes of the pulldown menubar. This resource is made available to allow applications to obtain the pulldown menubar Id, which is needed when attempting to add a pulldown menu to a widget which is not menu smart.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNshadowOn	XtCShadowOn	Boolean	TRUE
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground

Menu Manager Resource Set -- XWMENUMGR(3X)			
Name	**Class**	**Type**	**Default**
XtNassociateChildren	XtCAssociateChildren	Boolean	TRUE
XtNmenuPost	XtCMenuPost	String	"<Btn1Down>"
XtNmenuSelect	XtCMenuSelect	String	"<Btn1Up>"
XtNmenuUnpost	XtCMenuUnpost	String	NULL
XtNkbdSelect	XtCKbdSelect	String	"<Key>Select"

PULLDOWN BUTTON RESOURCES

The pulldown menu manager is responsible for managing the set of menupanes specified by the application, and for creating pulldown buttons within the pulldown menu bar, as needed. When creating the pulldown buttons, certain resources are inherited from the menupane from which the pulldown button is derived, while other resources are inherited from the menu manager. When an application modifies one of these resources within the menupane or the menu manager, the attribute will also be passed on to the associated pulldown button. The following tables outline those resources which are inherited from the menupane and those which are inherited from the menu manager:

Inherited MenuPane Resource Set			
Name	**Class**	**Type**	**Default**
XtNfont	XtCFont	XFontStruct *	"fixed"
XtNforeground	XtCForeground	Pixel	Black
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtopShadowColor	XtCBackground	Pixel	White
XtNtopShadowTile	XtCTopShadowTile	int	50_foreground
XtNbottomShadowColor	XtCForeground	Pixel	Black
XtNbottomShadowTile	XtCBottomShadowTile	int	foreground

Inherited Menu Manager Resource Set			
Name	**Class**	**Type**	**Default**
XtNshadowOn	XtCShadowOn	Boolean	TRUE

BUGS

Due to limitations within the Xt Intrinsics, keyboard accelerators for posting a menu pane or for selecting a menu item do not work if the widget to which the menu manager is attached has traversal enabled.

The pulldown menu manager currently does not support keyboard traversal.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWMENUMGR(3X)

NAME

XwrowColWidgetClass – the X Widgets row/column manager widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/RCManager.h>
```

CLASSES

The row column manager widget is built from the Core, Composite, Constraint and XwManager classes. Note that the Constraint fields are not used in this widget and so is not listed in the resource tables below. Also, since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating a row column manager is **XwrowColWidgetClass**. The class name is **RowCol**.

DESCRIPTION

The row/column widget is a composite widget which supports 3 types of row/column layouts for its children. They are: requested columns, maximum columns, and maximum unaligned. With the first layout type, requested columns, the application specifies the number of columns (the default is one) to be used in laying out the data. The children are laid out rowwise. Columns are as wide as the widest element in the column and all elements are left justified. Row height is determined by the largest element in the row and all elements are centered in the row. The second layout type, maximum columns, automatically calculates the maximum number of columns that can fit within the manager and lays the children out accordingly. The last layout type, maximum unaligned, does not force any columnar alignment. A child being positioned is placed to the immediately right of previous child until a row is full, then a new row is started at the left edge of the manager immediately below the previous row.

In addition to the row/column ordering, this manager widget supports 3 different layout policies: minimize (the default), maximize and ignore. When the layout policy is set to minimize, the manager will create a box which is just large enough to contain all of its children, regardless of any provided width and height values. When the given width and height values would create a box larger than needed, the maximize setting will use this additional space as padding between elements. Note that, with the maximize setting, if one or both of the height/width values are too small, the box will grow the manager to honor the given width and height, it will not grow or shrink in response to the addition, deletion or altering of its children.

The row/column widget also implements two selection policies. The default is n_of_many, and the alternative is one_of_many. The n_of_many policy does not require any action on the part of the manager widget. It allows any or all of its children to be selected and performs no action in response to their selection. The one_of_many policy ONLY applies to to children widgets which are subclasses of the XwPrimitive class. When one_of_many is the active policy, a callback (of type XtNselect) is added to each child widget. Then, when a child is selected the manager is informed. The manager keeps track of the previously active child and directly invokes a release procedure in that child so that it becomes unselected. The one_of_many mode will not activate a child if none are active and will not disallow the selection of an active child causing it to become deactive. Thus, if a strict one of many mode is desired, the application will have to enforce it.

NEW RESOURCES

The row/column manager defines a set of resource types used by the programmer to specify data for the manager widget. The programmer can also set the values for the Core, Composite and XwManager widget classes to set attributes for this widget. The following table contains the settable resources defined by the row/column manager. The string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNvSpace becomes vSpace.

Row Column Resource Set			
Name	Class	Type	Default
XtNhSpace	XtCHSpace	int	4
XtNvSpace	XtCVSpace	int	4
XtNlayoutType	XtCLayoutType	int	requested_columns
XtNcolumns	XtCColumns	int	1
XtNforceSize	XtCForceSize	Boolean	FALSE
XtNsingleRow	XtCSingleRow	Boolean	FALSE
XtNmode	XtCMode	int	n_of_many

XtNhSpace
The application may determine the number of pixels of space left between each element within a given row. This defines a minimum spacing.

XtNvSpace
The application may determine the number of pixels of space left between each column. This defines a minimum spacing.

XtNlayoutType
The application can specify the type of layout the row column manager is to perform. Allowable argument list settings are XwREQUESTED_COLUMNS, XwMAXIMUM_COLUMNS and XwMAXIMUM_UNALIGNED. To set this value in .Xdefaults or another resource file use the strings requested_columns, maximum_columns and maximum_unaligned.

XtNcolumns
The application can specify the number of columns to be used when laying out the widgets children.

XtNforceSize
The application has the option of forcing the widths of each widget in a column and the heights of each widget in a row to be the same. This can be used, for example to enforce an orderly layout for a group of buttons. For the layout type of maximum unaligned, only the heights of the widgets in a row are forced to the same size.

XtNsingleRow
For layout types of maximum columns and maximum unaligned, the application has the option of having the row column manager to try to lay itself out in a single row whenever one of its children makes a geometry request.

XtNmode
The application can specify whether the selection policy is n_of_many or one_of_many. Allowable argument list settings are XwONE_OF_MANY and XwN_OF_MANY. To set this value in .Xdefaults or another resource file use

the strings one_of_many and n_of_many.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWPRIMITIVE(3X)

NAME

XwRegisterConverters – register all of the resource converters used by the X Widgets.

SYNOPSIS

```
#include <X11/Atoms.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>

void XwRegisterConverters ()
```

DESCRIPTION

XwRegisterConverters is used by widget writers to register all of the resource type converters used by the X Widgets. The call to this routine is made within a widget's ClassInitialize procedure that has added a resource converter to the source file containing this function. XwRegisterConverters ensures that resource converters it is registering are only registered once.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

NAME

XwsashWidgetClass – an X Widgets utility widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Sash.h>
```

CLASSES

The sash widget is built from the Core and XwPrimitive classes.

The widget class to use when creating a sash is **XwsashWidgetClass**. The class name is **Sash.**

DESCRIPTION

The sash widget is a utility widget used by the vertical paned manager XwVPaned to control the sizes of the individual panes. In its realized form it appears as a square box of its background color. When the pointer is moved into the sash the cursor is changed to the crosshair cursor.

Callbacks can be attached to the widget to report selection (XtNselect) and unselection (XtNrelease). This widget can be set to respond to Enter and Leave window events by highlighting and unhighlighting the sash. This widget is also capable of handling keyboard traversal. (While you can traverse to the Sash in the current widget library, Sash does not handle keyboard input.) See the translations below for the default traversal keycodes.

NEW RESOURCES

The sash widget class defines one additional resource which is detailed in the table below. The programmer should refer to the man pages for the sash's superclasses to determine available resources and their defaults.

Sash Resource Set			
Name	Class	Type	Default
XtNcallback	XtCCallback	caddr_t	NULL

XtNcallback

This is used by the paned window widget to be informed of button presses and mouse movement associated with the sash.

INHERITED RESOURCES

The following resources are inherited from the named superclasses: The defaults used for the sash when being created are as follows:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHTLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the sash widget.

TRANSLATIONS

The input to the sash is driven by the mouse buttons. The default translation set defining this button is listed below. Note that for the specific key symbols used in traversal, the HP Key Cap which corresponds to this key symbol appears to the right of the definition.

<Btn1Down>:	SashAction(Start, UpperPane)
<Btn2Down>:	SashAction(Start, ThisBorderOnly)
<Btn3Down>:	SashAction(Start, LowerPane)
<Btn1Motion>:	SashAction(Move, Upper)
<Btn2Motion>:	SashAction(Move, ThisBorder)
<Btn3Motion>:	SashAction(Move, Lower)
Any<BtnUp>:	SashAction(Commit)
<EnterWindow>:	enter()
<LeaveWindow>:	leave()

ACTIONS

SashAction(Start, UpperPane):
Change the cursor from the crosshair to an upward pointing arrow. Determine the upper pane which will be adjusted (usually the pane to which the sash is attached).

SashAction(Start, ThisBorderOnly):
Change the cursor from the crosshair to a double headed arrow. The panes that will be adjusted are the pane to which the sash is attached and the first pane below it that can be adjusted. Unlike the UpperPane and LowerPane mode, only 2 panes will be affected. If one of the panes reaches its minimum or maximum, adjustment will stop, instead of finding the next adjustable pane.

SashAction(Start, LowerPane):
Change the cursor from the crosshair to a downward pointing arrow. Determine the lower pane which will be adjusted (usually the pane below the pane to which the sash is attached).

SashAction(Move, Upper):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget below the upper pane can be adjusted and make the appropriate adjustments.

SashAction(Move, ThisBorder):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Adjust as needed (and as possible) the upper and lower panes selected when the SashAction(Start, ThisBorderOnly) action was invoked.

SashAction(Move, Lower):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget above the lower pane can be adjusted and make the appropriate adjustments.

enter:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the button will be highlighted. Otherwise no action is taken.

leave:
If the XtNtraversalType resouces has been set to XwHIGHLIGHT_ENTER then the button will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWVPANED(3X)

NAME

XwscrollbarWidgetClass – the X Widget's scrollbar widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Valuator.h>
#include <Xw/Arrow.h>
#include <Xw/ScrollBar.h>
```

CLASSES

The ScrollBar is built from the Core, Composite, and XwManager classes.

The widget class to use when creating a scrollbar is **XwscrollbarWidgetClass**. The class name for scrollbar is **ScrollBar.**

DESCRIPTION

The ScrollBar widget combines the Valuator and Arrow widgets to implement a horizontal or vertical scrolling widget containing a valuator and an arrow on each end of the valuator.

As with the Valuator, input is supported through interactive slider movement and selections on the slide area not occupied by the slider. Both types of input have a separate callback list for communicating with the application. The arrows on each end of the valuator control additional input to the valuator. When an arrow is selected, the slider within the valuator will be moved in the direction of the arrow by an application supplied amount. If the button is held down, the slider will continue to move at a constant rate.

The ScrollBar can be used by the application to attach to objects scrolled under application control, or used by composite widgets to implement predefined scrolled objects.

NEW RESOURCES

The ScrollBar defines a set of resource types used by the programmer to specify the data for the scrollbar. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by ScrollBar.

ScrollBar Resource Set			
Name	**Class**	**Type**	**Default**
XtNinitialDelay	XtCinitialDelay	int	500
XtNrepeatRate	XtCRepeatRate	int	100
XtNgranularity	XtCGranularity	int	2

XtNinitialDelay

The ScrollBar supports smooth time sequenced movement of the slider when a selection occurs on the arrows. This resource defines the amount of delay to wait between the initial selection and the slider starting its repetitive movement. The value is defined in milliseconds.

XtNrepeatRate
This resource defines the continuous repeat rate to use to move the slider while the button is being held down on an arrow. The value is also defined in milliseconds.

XtNgranularity
This resource defines the increment in the valuator's coordinate system to move the slider while continuous scrolling.

INCORPORATED RESOURCES

The ScrollBar creates itself by internally creating two Arrow widgets and a Valuator. As such, it uses a large number of the resources defined by these widgets. Many of the attributes for these widgets can be set through the .Xdefaults file or by use of XtSetValues() when communicating with the ScrollBar.

It should be noted, that only the resources within the following tables will have any effect on the valuator or arrows. The other resource types defined by the Valuator and Arrow widgets are either overridden or unused by ScrollBar.

The following tables list the resources incorporated by ScrollBar. For a complete description of these resources, refer to the manual page listed in the table heading.

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNtraversalType	XtCTraversalType	int	highlight_off

Valuator Resource Set -- XWVALUATOR(3X)			
Name	**Class**	**Type**	**Default**
XtNsliderMin	XtCSliderMin	int	0
XtNsliderMax	XtCSliderMax	int	100
XtNsliderExtent	XtCSliderExtent	int	10
XtNsliderOrigin	XtCSliderOrigin	int	0
XtNslideOrientation	XtCSlideOrientation	int	vertical
XtNsliderMoved	XtCCallback	Pointer	NULL
XtNsliderReleased	XtCCallback	Pointer	NULL
XtNareaSelected	XtCCallback	Pointer	NULL

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to True at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWPRIMITIVE(3X), XWCREATETILE(3X), XWVALUATOR(3X), XWARROW(3X)

NAME

XwswindowWidgetClass – the X Widget's scrolled window widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Valuator.h>
#include <Xw/Arrow.h>
#include <Xw/ScrollBar.h>
#include <Xw/SWindow.h>
```

CLASSES

The ScrolledWindow is built from the Core, Composite, and XwManager classes.

The widget class to use when creating a scrolled window is **XwswindowWidgetClass.** The class name is **ScrolledWindow.**

DESCRIPTION

The ScrolledWindow widget combines the ScrollBar and BulletinBoard widgets to implement a visible window onto some other (usually larger) data display. The visible part of the window can be scrolled through the larger display by the use of scroll bars.

To use the scrolled window, an application first creates a ScrolledWindow widget, and then creates a widget capable of displaying the desired data as a child of the ScrolledWindow. ScrolledWindow will position the child widget within its BulletinBoard manager instance, and create scroll bars for the horizontal and vertical dimensions. When the user performs some action on the scroll bars, the child widget will be repositioned accordingly within the bulletin board.

NEW RESOURCES

The ScrolledWindow widget defines a unique set of resource types which can be used by the programmer to control the appearence and behavior of the scrolled window. The programmer can also set the values for the Core, Composite and Manager widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by ScrolledWindow.

ScrolledWindow Resource Set			
Name	Class	Type	Default
XtNvsbWidth	XtCVsbWidth	int	20
XtNhsbHeight	XtCHsbHeight	int	20
XtNforceHorizontalSB	XtCForceHorizontalSB	Boolean	FALSE
XtNforceVerticalSB	XtCForceVerticalSB	Boolean	FALSE
XtNvScrollEvent	XtCCallBack	Pointer	NULL
XtNhScrollEvent	XtCCallBack	Pointer	NULL
XtNinitialX	XtCInitialX	int	0
XtNinitialY	XtCInitialY	int	0

XtNvScrollBarWidth
This is the width in pixels of the vertical scroll bar.

XtNhScrollBarHeight
This is the height in pixels of the horizontal scroll bar.

XtNforceHorizontalSB
When the child widget is created and positioned within the scrolled window, its width and height are examined. If the entire child widget will fit within the width of the scrolled window, the horizontal scrollbar will not be created, since there is no need to scroll in that direction. Setting this resource to TRUE disables this checking and will force a horizontal scrollbar to be attached to the window regardless of the dimension of the child widget.

XtNforceVerticalSB
This resource controls the existence of the vertical scrollbar. As described above, if this is set to TRUE a vertical scrollbar will always be created.

XtNvScrollEvent and XtNvScrollEvent
An application program may track the position of the child within the scrolled window by linking into these callbacks. Whenever the user moves the valuator in either scroll bar, ScrolledWindow moves the child accordingly and then calls the appropriate callback. The call_data parameter is set to the new valuator origin for the scrollbar.

XtNinitialX and XtNinitialY
The child widget is initially positioned at (0,0) within the bulletin board. This positioning can be changed by specifying a new X and Y location. If a non-zero value is given, that becomes the initial location, and the valuators inside the scrollbars are adjusted to give a visual indication of the new offset. **This value should be negative to assure proper operation of the scrolled window.** These resources are only used at initialization time; they cannot be set through a call to XtSetValues.

INCORPORATED RESOURCES

The ScrolledWindow widget is built from two ScrollBar widgets and a BulletinBoard widget. As such, it uses a large number of the resources defined by these widgets. Many of the attributes for these widgets can be set through the .Xdefaults file or by use of XtSetValues() when communicating with the ScrolledWindow widget.

Only the resources within the following tables will have any effect on the scroll bars. The other resource types defined by the ScrollBar widget are either overridden or unused by ScrolledWindow.

The following tables list the resources incorporated by ScrolledWindow. For a complete description of these resources, refer to the manual page listed in the table heading.

ScrollBar Resource Set -- XWSCROLLBAR(3X)			
Name	Class	Type	Default
XtNinitialDelay	XtCinitialDelay	int	500
XtNrepeatRate	XtCRepeatRate	int	100
XtNgranularity	XtCGranularity	int	10
XtNforeground	XtCForeground	Pixel	Black
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightTile	XtCHighlightTile	int	50%_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalType	XtCTraversalType	int	highlight_off

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the scrolled window widget.

TRANSLATIONS

Input to the ScrolledWindow widget is driven by the mouse buttons. However the translations driving the actions are defined by the ScrollBar widgets. The additional translations used for ScrolledWindow are as follows:

```
<EnterWindow>:      enter(),
<LeaveWindow>:      leave(),
```

ACTIONS

enter: Enter window events occurring on the scrolled window are handled by this action.

leave: Leave window events occurring on the scrolled window are handled by this action.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X) XWPRIMITIVE(3X), XWSCROLLBAR(3X),XWBULLETINBOARD(3X),XWVALUATOR(3X), XWARROW(3X)

NAME

XwstaticrasterWidgetClass – The HP X Widget's static image widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/SRaster.h>
```

CLASSES

The static raster widget is built from the Core, XwPrimitive and XwSRaster classes.

The widget class to use when creating a static raster is **XwstaticrasterWidgetClass.** The class name is **StaticRaster.**

DESCRIPTION

The static raster widget provides an uneditable raster image. As a default, the image is placed in a window that is exactly the size of the raster (plus the border width). The image can be dynamically resized. If the window is enlarged from its original size, the image will be redrawn in the center of the new window. If the window shrinks below the size of the raster, the image is clipped on the right and bottom sides as needed to fit within the new boundries.

The raster image is provided to the widget in the form of an XImage data structure. New data can be displayed by specifing a new XImage structure, or by changing the pointer to the bitmap data within that structure.

Callbacks can be attached to the widget to report selection (XtNselect) and unselection (XtNrelease). This widget can be set to respond to Enter and Leave window events by highlighting and unhighlighting the border.

NEW RESOURCES

StaticRaster defines several new resources. (To reference a resource in a .Xdefaults file, strip off the XtN from the resource string.)

StaticRaster Resource Set			
Resource	**Class**	**Type**	**Default**
XtNsRimage	XtCSRimage	XImage *	NULL
XtNinvertOnSelect	XtCInvertOnSelect	Boolean	TRUE
XtNshowSelected	XtCIShowSelected	Boolean	TRUE
XtNset	XtCSet	Boolean	FALSE

XtNsRimage
This is a pointer to an XImage data structure.

XtNinvertOnSelect
If this resource is TRUE, the raster image will invert its foreground and background colors when selected, and return to normal when unselected.

XtNshowSelected
If TRUE, this will cause the image to appear to be indented when selected, and raised when unselected.

XtNset
This is a Boolean resource which indicates whether the raster is currently selected (TRUE) or not (FALSE).

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. See the TRANSLATIONS section in this man page for a description of the translations local to the static raster widget.

TRANSLATIONS

The static raster is affected by the mouse buttons and cursor motion. The default translation set is as follows:

```
<Btn1Down>:     select(),
<Btn1Up>:       release(),
<EnterWindow>:  enter(),
<LeaveWindow>:  leave(),
```

ACTIONS

select:
Allows an application to be notified of the event via the callback structure.

release:
Allows an application to be notified of the event via the callback structure.

enter:
Causes the border to be highlighted if enabled.

leave:
Causes the border to be highlighted if enabled.

NOTES

Error checking on the XImage structure is minimal, so weird rasters can result from incorrect or incomplete data.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X)

NAME

XwstatictextWidgetClass – An X Widget for displaying static text.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/SText.h>
```

CLASSES

A subclass of CoreClass and XwPrimitiveClass.

The widget class to use when creating a static text widget is **XwstatictextWidgetClass.**

The class name for StaticText is **StaticText.**

DESCRIPTION

StaticText provides an uneditable block of text. Optionally StaticText will provide simple heuristics to fit the text into arbitrarily sized windows. Imbedded new-line characters in the string are always honored. Stripping of leading and trailing spaces is optional.

If the static text widget is directed to become larger than is needed for the text, the text will be centered in the window. The text will retain the specified alignment.

If the static text widget is directed to become narrower than is neccessary for the text, the text may be wrapped (depending on XtNWrap) or clipped to the right and/or left (depending on the XtNalignment).

If the static text widget is directed to become shorter than is neccessary for the text, the text will be clipped on the bottom.

When the text is wrapped, StaticText will try to break lines on spaces. The space on which the line is broken is temporarily converted to a newline.

NEW RESOURCES

To specify any of these resources within a resource defaults file, simply drop the *XtN* prefis from the resource name. StaticText defines the following new resources:

StaticText Resource Set			
Name	**Class**	**Type**	**Default**
XtNhSpace	XtCHSpace	int	2
XtNvSpace	XtCVSpace	int	2
XtNalignment	XtCAlignment	XwAlignment	XwALIGN_LEFT
XtNgravity	XtCGravity	int	CenterGravity
XtNwrap	XtCWrap	Boolean	TRUE
XtNstrip	XtCStrip	Boolean	TRUE
XtNlineSpace	XtCLineSpace	int	0
XtNfont	XtCFont	XFontStruct *	Fixed
XtNstring	XtCString	char *	NULL

XtNhSpace

This specifies the number of pixels to maintain between the text and the highlight area to the right and left of the text.

XtNvSpace
This specifies the number of pixels to maintain between the text and the highlight area to the top and bottom of the text.

XtNalignment
This specifies the alignment to be applied when drawing the text. The alignment resource is interpreted without regard to case.

Alignment never causes leading or trailing spaces to be stripped.

Alignment may have the following values and effects:

XwALIGN_LEFT will cause the left sides of the lines will be vertically aligned. Specified in resource default file as "Left".

XwALIGN_CENTER will cause the centers of the lines will be vertically aligned. Specified in resource default file as "Center".

XwALIGN_RIGHT will cause the right sides of the lines will be vertically aligned. Specified in resource default file as "Right".

XtNgravity
This resource controls the use of extra space within the widget.

CenterGravity will cause the string to be centered in the extra space. Specified in the resource defaults file as "CenterGravity".

NorthGravity will cause the string to always to be at the top of the window centered in any extra width. Specified in the resource defaults file as "NorthGravity".

SouthGravity will cause the string to always to be at the bottom of the window centered in any extra width. Specified in the resource defaults file as "SouthGravity".

EastGravity will cause the string to always be at the right of the window centered in any extra height. Specified in the resource defaults file as "EastGravity".

WestGravity will cause the string to always be at the left of the window centered in any extra height. Specified in the resource defaults file as "WestGravity".

NorthWestGravity will cause the string to always be in the upper left corner of the window. Specified in the resource defaults file as "NorthWestGravity".

NorthEastGravity will cause the string to always be in the upper right corner of the window. Specified in the resource defaults file as "NorthEastGravity".

SouthWestGravity will cause the string to always be in the lower left corner of the window. Specified in the resource defaults file as "SouthWestGravity".

SouthEastGravity will cause the string to always be in the lower

right corner of the window. Specified in the resource defaults file as "SouthEastGravity".

XtNwrap

This resource controls the wrapping of lines within the widget. If XtNwrap is TRUE, lines which are too long are broken on spaces. The spaces are converted to new-lines to break the line. Imbedded new-lines are honored. If there is too much text for the specified window size, it will be clipped at the bottom.

If XtNwrap is FALSE, lines which are too long will be clipped according to the alignment. An XtNalignment value of XwALIGN_LEFT will cause lines which are too long to be clipped to the right. An XtNalignment value of XwALIGN_RIGHT will cause lines which are too long to be clipped to the left. An XtNalignment value of XwALIGN_CENTER will cause lines to be clipped equally on both the right and the left.

XtNstrip

This resource controls the stripping of leading an trailing spaces during the layout of the text string. If XtNstrip is FALSE, spaces are not stripped. If XtNstrip is TRUE and XtNalignment is XwALIGN_LEFT, leading spaces are stripped from each line. If XtNstrip is TRUE and XtNalignment is XwALIGN_CENTER, both leading and trailing spaces are stripped from each line. If XtNstrip is TRUE and XtNalignment is XwALIGN_RIGHT, trailing spaces are stripped from each line.

XtNlineSpace

This resource controls the amount of space between lines. It is specified as a percentage of the font height. This space is added between each line of text. XtNlineSpace may be negative to a maximum of -100 (which causes all lines to overwrite each other).

XtNfont

This resource controls which font the text will drawn in.

XtNstring

This resource is the string which will be drawn. The string must be null terminated. If the string is given in a resource defaults file, newlines may be specified by "\n" within the string.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

TRANSLATIONS

The input to the toggle is driven by the mouse buttons. The default translation set defining this button is listed below. Note that for the specific key symbols used in traversal, the HP Key Cap which corresponds to this key symbol appears to the right of the definition.

<EnterWindow>:	enter()	
<LeaveWindow>:	leave()	
<KeyDown>Select:	select()	HP "Select" key
<KeyUp>Select:	release()	HP "Select" key

ACTIONS

enter

If the XtNtraversalType resource has been set to XwHIGHLIGHT_OFF then the StaticText will be highlighted. Otherwise no action is taken.

leave

If the XtNtraversalType resouces has been set to XwHIGHLIGHT_OFF then the StaticText will be unhighlighted. Otherwise no action is taken.

select

Invokes the select callbacks.

release

Invokes the release callbacks.

NOTES

The forced new line is the '\n' character constant as defined by the C compiler. Fonts which do not use that character constant for the newline will not be handled correctly by StaticText.

StaticText will assume that the space is the ' ' character constant as defined by the C compiler. Fonts which do not use that character constant for spaces will not be handled correctly by StaticText.

Non-8-bit character representations have undefined effects on the operation of StaticText.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

XWPRIMITIVE(3X)

NAME

XwtexteditWidgetClass – An X Widget for viewing and editing text.

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/TextEdit.h>

CLASSES

TextEdit is a subclass of CoreClass and XwPrimitiveClass.

The widget class record to use when creating a text edit widget is **XwtexteditWidgetClass.**

The class name for TextEdit is **TextEdit.**

OVERVIEW

TextEdit provides a single and multi-line text editor which has both a customizable user interface and a programmatic interface. It can be used for single-line string entry, forms entry with verification procedures, multiple-page document viewing, and full-window editing. It provides an application with a consistent editing paradigm for entry of textual data.

The display of the textual data on the screen can be adjusted to the application writer's need based on four class resources, XtNwrap, XtNwrapBreak, XtNscroll, and XtNgrow. XtNwrapP controls automatic line breaking for lines that extend off the end of the screen. XtNscroll controls horizontal and vertical shifting of the text when the insertion cursor moves off the screen. XtNgrow controls attempts by the widget to resize its window to make more room for text that extends beyond the current screen size. These resources are explained in detail below.

TextEdit provides separate callback lists to verify insertion cursor movement, modification of the text, and leaving the TextEdit widget. Each of these callbacks provides the verification function with the widget instance, the event that caused the callback, and a data structure specific to the verification type. From this information the function can verify if the application considers this to be a legitimate state change and signal the widget whether to continue with the action. The verification function can also manipulate the widget through the class methods defined by the TextEdit class. The verification callback lists are explained in detail below.

The user interface can be tailored by providing a new set of translations. The default translations provide commands for movement, deletion, killing and selection with key bindings similar to an EMACS style editor.

TextEdit allows the user to select regions of text. By using TextEdit's selection mechanism, application writers can easily fit instances of TextEdit into X11's current selection mechanism.

The TextEdit class controls the data structures for drawing the text on the screen and defines the functions that manipulate that data. The storage of the text is provided by a separate component called the Source. The Source provides the storage of the textual data and a set of functions for querying and changing that data. The application writer can provide a new source for the TextEdit widget. The details are provided below.

NEW RESOURCES

TextEdit defines the following new resources:

TextEdit Resource Set			
Name	Class	Type	Default
XtNsourceType	XtCSourceType	String	"stringsrc"
XtNsource	XtCTextSource	Pointer	StringSrc
XtNdisplayPosition	XtCTextPosition	XtTextPosition	0
XtNinsertPosition	XtCTextPosition	XtTextPosition	0
XtNselectionLeft	XtCSelectionLeft	XtTextPosition	0
XtNselectionRight	XtCSelectionRight	XtTextPosition	0
XtNwrap	XtCWrap	XwWrap	XwWrapOff
XtNwrapBreak	XtCWrapBreak	XwWrapBreak	XwWrapWhiteSpace
XtNscroll	XtCScroll	XwScroll	XwAutoScrollOff
XtNgrow	XtCGrow	XwGrow	XwGrowOff
XtNleftMargin	XtCMargin	Dimension	3 See Note Below
XtNrightMargin	XtCMargin	Dimension	3 See Note Below
XtNtopMargin	XtCMargin	Dimension	3 See Note Below
XtNbottomMargin	XtCMargin	Dimension	3 See Note Below
XtNmotionVerification	XtCCallback	XtRCallback	NULL
XtNmodifyVerification	XtCCallback	XtRCallback	NULL
XtNleaveVerification	XtCCallback	XtRCallback	NULL
XtNexecute	XtCallback	XtRCallback	NULL

XtNtranslations
The set of default translations are described below.

XtNdisplayPosition
The position in the text source that will be displayed at the top of the screen. The default is 0, or the start of the text source.

XtNinsertPosition
The position in the text source of the insert cursor. The default is 0.

XtNselectionLeft
The starting position of the initial selection. The default is 0.

XtNselectionRight
The ending position of the initial selection. The default is 0.

XtNsourceType
This defines the type of the text source. It is one of "stringsrc," "disksrc" or "progdefinedsource."

XtNsource
This specifies a new Source. The default is StringSrc.

XtNwrap
This resource specifies how the widget displays lines longer than the screen width. When set to XwWrapOff, the lines may extend off screen to the right. When set to XwSoftWrap, the lines will be wrapped at the right margin with the actual position determined by the resource XtNwrapBreak.

XtNwrapBreak
This resource specifies how the wrap position is determined. When set to XwWrapAny, the wrap will happen at the character position closest to the right margin. When set to XwWrapWhiteSpace, the wrap will happen at the last whitespace before the right margin. If the line does not have whitespace, it will be wrapped as XwWrapAny.

XtNscroll
This resource controls the horizontal and vertical scrolling of lines longer than the screen width. When set to XwAutoScrollOff the widget will not scroll. When set to XwAutoScrollVertical, the widget will scroll lines vertically. When set to XwAutoScrollHorizontal, the widget will scroll a single-line display horizontally. Horizontal scrolling is not currently supported for multi-line displays. Both horizontal and vertical scrolling can be set with XwAutoScrollBoth (again, subject to the single-line horizontal restriction). The default is XwAutoScrollOff. XtNscroll has lower priority than XtNwrap, meaning if wrapping is enabled, the widget will attempt to wrap to the next line before it will attempt to scroll horizontally.

XtNgrow
This resource controls if the widget will try to resize its window when it needs more height or width to display the text. When set to XwGrowOff it will not resize itself. When set to XwGrowHorizontal it will attempt to change its width when lines are too long for the current screen width. When set to XwGrowVertical it will attempt to resize its height when the number of text lines is greater than can be displayed with the current screen height. When set to XwGrowBoth, the widget will attempt resizes in both dimensions. Growth attempts have higher priority than either wrapping or scrolling. If enabled, the widget will always try to grow to display text before trying to wrap or scroll. The default is XwGrowOff. The success of a resize request is determined by the widget's parent.

XtNleftMargin
The number of pixels used for the left margin.

NOTE: if TextEdit is embedded in a manager with keyboard traversal enabled, it will silently enforce the constraint that all margins must be at least 3 pixels wider than the highlight border width.

XtNrightMargin
The number of pixels used for the right margin.

XtNtopMargin
The number of pixels used for the top margin.

XtNbottomMargin
The number of pixels used for the bottom margin.

XtNmotionVerification
This verification callback list is called before the insertion cursor is moved to a new position. The default is NULL. See the verification section below.

XtNmodifyVerification
This verification callback list is called before text is deleted from or inserted to the text source. The default is NULL. See the verification section below.

XtNleaveVerification
This verification callback list is called before the widget loses input focus. The default is NULL. See the verification section below.

XtNexecute
This callback list is similar to a selection function on a button. When the user invokes an event that calls the "execute" function (see the translation table below), this callback list will be executed. In the default translation table, this is bound to the "enter" key.

XWTEXTEDIT(3Xh)

SUBCOMPONENT RESOURCES

StringSrc defines the following new resources. In a resource file they can be specified by the name stringsrc under the name of the TextEdit widget, or through the class StringSrc.

StringSrc Resource Set			
Name	Class	Type	Default
XtNstring	XtCString	char *	NULL
XtNmaximumSize	XtCLength	int	NULL
XtNeditType	XtCEditType	XtEditType	XwtextEdit

XtNstring
The initial string to be viewed and/or edited. The default is the empty string. An XtGetValues call on this resource will return a copy of the internal buffer. The application program is responsible for freeing the space allocated by the copy. An XtSetValues call will copy the given string into the internal buffer.

XtNmaximumSize
The maximum number of characters that can be entered into the internal buffer. If this value is not set then the internal buffer will increase its size as needed limited only by the space limitations of the process.

XtNeditType
This resource controls the edit state of the source. It can be XttextRead, a read only source, XttextAppend, a source than can only be appended to, and XttextEdit, a fully editable source.

DiskSrc defines the following new resources. In a resource file they can be specified by the name disksrc under the name of the TextEdit widget, or through the class DiskSrc.

DiskSrc Resource Set			
Name	Class	Type	Default
XtNfile	XtCFile	char *	NULL
XtNeditType	XtCEditType	XtEditType	XwtextEdit

XtNfile
The absolute pathname of a disk file to be viewed and/or edited. If no file is given, a temporary file will be created.

XtNeditType
This resource controls the edit state of the source. It can be XttextRead, a read only source, and XttextAppend, a source than can only be appended to.

Display defines the following new resources. In a resource file they can be specified by the name display under the name of the TextEdit widget, or through the class Display.

Display resource Set			
Name	Class	Type	Default
XtNfont	XtCFont	XFontStruct *	Fixed
XtNforeground	XtCForeground	XtRPixel	Black

XtNfont
The font used to display the text. The default is fixed. There are currently several display bugs associated with proportional fonts.

XtNforeground
The color for drawing the text. The default is black.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the scrolled window widget.

TRANSLATIONS

Since TextEdit has full editing functionality, it supports an elaborate set of translations. The following table lists TextEdit's default translations which are a subset of key bindings from an EMACS editor. (An EMACS editor refers to a set of editors based on the original design of R.M. Stallman at MIT for an extensible, customizable self-documenting display editor.) TextEdit supports the concept of delete and kill. Both delete and kill remove a unit of text from the text source, but text that has been removed with a kill can be restored by an unkill action. Kills are stored in the X Cutbuffer 1, so that a kill in one instance of a TextEdit widget can be inserted into another instance of a TextEdit widget. TextEdit does not support a history of kills in a kill ring, nor the appending of kills made in sequence. TextEdit highlights the current selection by reversing the foreground and background color. Text that has been copied from TextEdit into the current selection storage can be inserted into the buffer with a stuff action.

Each of these functions can be rebound to a different key in the default translation file set in .Xdefaults. The string to identify the function is identical to the function name used below. An example line in that file to bind Control-I to move the insertion point forward one word is:

Ctrl<Key>I: forward-word

See the Xt Intrinsics documention for more information on the Xdefaults file and translations.

TextEdit works with keyboard traversal and defines the required actions.

DEFAULT KEY BINDINGS FOR TEXTEDIT

Movement	
Ctrl F	forward-character
Right Arrow	forward-character
Ctrl B	backward-character
Left Arrow	backward-character
Meta F	forward-word
Meta B	backward-word
Meta]	forward-paragraph
Ctrl [	backward-paragraph
Ctrl A	beginning-of-line
Ctrl E	end-of-line
Ctrl N	next-line
Down Arrow	next-line
Ctrl P	previous-line
Up Arrow	previous-line
Ctrl V	next-page
Next	next-page
Meta V	previous-page
Prev	previous-page
Meta <	beginning-of-file
Home	beginning-of-file
Meta >	end-of-file
Shift Home	end-of-file
Ctrl Z	scroll-one-line-up
Meta Z	scroll-one-line-down

Delete Kill and Stuff	
Ctrl D	delete-next-character
Ctrl H	delete-previous-character
Meta D:	delete-next-word
Meta H	delete-previous-word
Shift Meta D	kill-word
Shift Meta H	backward-kill-word
Ctrl W	kill-selection
Ctrl K	kill-to-end-of-line
Meta K	kill-to-end-of-paragraph
Ctrl Y	unkill
Meta Y	stuff

Miscellaneous	
Ctrl J	newline-and-indent
Ctrl O	newline-and-backup
Ctrl M	newline
<Btn1Down>	select-start
Button1<PtrMoved>	extend-adjust
<Btn1Up>	extend-end
<Btn2Down>	stuff
<Btn3Down>	extend-start
Button3<PtrMoved>	extend-adjust
<Btn3Up>	extend-end
Ctrl L	redraw-display
<Key>	insert-char

KEYBOARD TRAVERSAL

The following table summarizes the keystrokes which (when keyboard traversal is active) will move the focus. The keys used elsewhere in the X Widgets library for keyboard traversal are used for other purposes in the text edit widget. Therefore, it was necessary to define other keystokes to serve these functions. To minimize the incompatibility the decision was made to use the same keys with the addition of the Ctl modifier to implement keyboard traversal in this widget.

Keyboard Traversal	
Ctrl Up	traverse-up
Ctrl Down	traverse-down
Ctrl Left	traverse-left
Ctrl Right	traverse-right
Ctrl Next	traverse-next
Ctrl Prev	traverse-prev
Ctrl Home	traverse-home
Enter	traverse-next-top

traverse-up:
Inform the parent of this widget that it should transfer keyboard focus to the first widget above this one.

traverse-down:
Inform the parent of this widget that it should transfer keyboard focus to the first widget below this one.

traverse-left:
Inform the parent of this widget that it should transfer keyboard focus to the first widget to the left of this one.

traverse-right:
Inform the parent of this widget that it should transfer keyboard focus to the first widget to the right of this one.

traverse-next:
Inform the parent of this widget that it should transfer keyboard focus to the next child in the parent's list of children.

traverse-prev:
Inform the parent of this widget that it should transfer keyboard focus to the previous child in the parent's list of children.

traverse-home:
Inform the parent of this widget that it should transfer keyboard focus to the child which is closest to the uppper left hand corner of the parent. If that child already has the keyboard focus, then ask the grandparent of this widget to give the keyboard focus to whichever of its children which is closest to the upper left hand corner.

traverse-next-top:
Find the topmost parent in this widget hierarch which is a subclass of XwManager and tell it to issues any XtNnextTop callbacks that have been registered with it. The purpose of this callback is to allow applications to move the keyboard focus between top level widget hierarchies of the same application.

DISPLAYING TEXT, WORD WRAP AND ACTIONS

Text is considered to be hierarchically composed of white space, words, lines and paragraphs. These component concepts are currently hard-coded, but we intend that future versions will support a more general version of the text composition hierarchy. White space is defined as any non-empty sequence of the ASCII characters space, tab, linefeed or carriage return (decimal values of 32, 9, 10, 13, respectively); a word is any non-empty sequence of characters bounded on both sides by whitespace. A source line is any (possibly empty) sequence of characters bounded by newline characters; a display line is any (possibly empty) sequence of characters appearing on a single screen display line. A source paragraph is any sequence of characters bounded by sets of two or more adjacent newline characters. a display paragraph is any (possibly empty) sequence of characters bounded by newline characters (NOTE: this is identical to the definition of a source line). In all cases, the beginning or end of the edit text is an acceptable bounding element in the previous definitions.

When making display decisions, TextEdit first determines whether all the text will fit in the current display. If it does not, and growing is enabled, the widget will make resize request of its parent. If the request is denied or only partially satisfied, no future growth requests will be made unless there is an intervening resize operation externally imposed. If any source line is still too long to fit in the display after growing is attempted, wrapping is checked. If wrap is off (XwWrapOff), one display line is drawn for each source line. If a source line is too long for the display, it is truncated at the right margin after the last full character which fits. If wrapping is enabled (XwSoftWrap), a new display line will be started with the first word which doesn't fit on the current line. If the wrap break option is XwWrapAny, as many characters from that word as will fit before the right margin are written to the current display line, then the next character starts at the left margin of the next display line. If the wrap break option is XwWrapWhiteSpace, the line break is instead made after the first whitespace character which follows the last full word which does fit on the current display line. If, however, under white space break, the first full word which does not fit is also the first word on the line, the wrap break is made as if XwWrapAny were selected.

VERIFICATION CALLBACKS

Three types of verification callbacks are supported by TextEdit There is one for motion operations, to verify a new insert position; there is one for modifying

operations, to verify insertion, deletion or replacement of text; there is one for widget exit, to verify state consistency on loss of focus by the widget. Each verification callback procedure is of type XtCallbackProc, which defines the three arguments it will be invoked with. These are the id of the widget making the callback, the client data which was specified by the client application when the callback was registered (see XtAddCallback), and a pointer (type XwTextVerifyPtr) to the verification call_data structure. The C data types used here are:

```
typedef enum {motionVerify, modVerify, leaveVerify} opType ;

typedef struct {
    XEvent      *xevent ;
    opType       operation ;
    boolean      doit ;
    XtTextPosition  currInsert, newInsert ;
    XtTextPosition  startPos, endPos ;
    XtTextBlock   *text ;
    } XwTextVerifyCD, *XwTextVerifyPtr ;
```

Before the chain of verification callbacks is activated for any given operation, a structure of type XtTextVerifyCD is initialized. The initial values are:

xevent:	for a leave operation, the current event pointer
operation:	element of opType signifying the type of verification operation
doit:	TRUE
currInsert:	current position of the insert point
newInsert:	for a motion operation, the position the user is attempting to move the insert point to, otherwise, the same value as currInsert
startPos:	for a modify operation, the beginning position in the current source of the text about to be deleted or replaced, or where new text will be inserted. If not a modify operation, the same value as currInsert.
endPos:	for a modify operation, the ending position in the current source of the text about to be deleted or replaced. If no text is being removed, it will have the same value as startPos. If not a modify operation, the same value as currInsert.
text:	for a modify operation with new text to be inserted, a pointer to a structure of type XtTextBlock, which references the text to be inserted. Otherwise, NULL.

It is possible for the client to register more than one callback procedure for any of these callback types. The order in which the callbacks will be invoked is described in the toolkit documentation. Since there can be more than one callback, each verification procedure should first check the doit field. If it is false,

someone else has already rejected the operation, so there is no need for further evaluation. On return from invoking the chain of callbacks, the TextEdit widget will look at the doit member of the XtTextVerifyCD structure. If it is still true, TextEdit will proceed with operation, otherwise it will not. Any user feedback for the rejected operation is the responsibility of the verification procedure. Verification callbacks are permitted to modify some of the data in the XtTextVerifyCD structure. The TextEdit widget will only look at certain fields on return, though, according to the operation type. For a motion operation, only the newInsert position will be looked at. For a modify operation, only startPos, endPos and text will be examined for changes. For leave operation, no fields will be examined. There is no mechanism for preventing a verification callback from making other changes to the editing state through the documented interface, but such behind-the-back actions are discouraged.

APPLICATION WRITER'S INTERFFACE

The state of TextEdit can be changed in through the normal functional interface to widgets (XtSetValues and XtGetValues) or by exported external functions.

TextEdit's resources can be queried and set through XtSetValues and XtGetValues. The widget will maintain its display consistent with the new values. In particular this is the method for changing the display options.

The internal buffer should be manipulated through the external functions that follow.

This set of external functions is designed to allow the widget programmer to access the internal buffer that TextEdit manages. For example, if the widget is being used to enter a string, the program can get a copy of the string (i.e. the internal buffer) with the function XwTextCopyBuffer or XwTextReadSubString. All of the following functions that change the contents of the buffer, its selection, or insertion position, will update the display after they are called. If the programmer needs to make a sequence of these calls, the widget's screen updating function should be turned off with a call to XwTextUpdate(Off) to prevent screen flash. After the sequence of calls the programmer must remember to call XwTextUpdate(On) to update the window and resume normal updating. Note that it is not necessary to turn off the update function for functions that only get values from the widget. Neither is it necessary to use these calls if the programmer only makes one call that changes the widget.

Buffer Functions

```
void XwTextClearBuffer(w)
        XwTextEditWidget w;
```

Clear the internal buffer. After this call all characters in the buffer have been removed.

```
unsigned char *XwTextCopyBuffer(w)
        XwTextEditWidget w;
```

This function uses XtMalloc to create space to make a copy of the internal buffer and returns the pointer to that copy. The application writer is responsible for freeing the space.

Read a Substring

```
int XwTextReadSubString( w, startpos, endpos, target, targetsize, targetused )
 XwTextEditWidget  w;
 XwTextPosition    startpos, endpos;
 unsigned char     *target;
 int               targetsize,
```

XWTEXTEDIT(3Xh)

*targetused;

This function will move characters from the buffer into the caller's space. The caller must provide the space to copy into and its size in bytes. The routine will return the number of positions moved. The value of targetused returns the number of bytes used in the target string by the move.

Selection

unsigned char *XwTextCopySelection(w)
XwTextEditWidget w;

This function uses XtMalloc to create space to make a copy of the current selection and returns the pointer to that copy. The application writer is responsible for freeing the space.

void XwTextUnsetSelection(w)
XwTextEditWidget w;

This function will clear the current selection.

void XwTextSetSelection(w, left, right)
XwTextEditWidget w;
XwTextPosition left, right;

This function sets the current selection to be between the character positions left to right.

void XwTextGetSelectionPos(w, left, right)
XwTextEditWidget w;
XwTextPosition *left, *right;

This function returns the character positions of the current selection.

Insertion and Deletion

void XwTextInsert(w, string)
XwTextEditWidget w;
unsigned char *string;

This function inserts the string at the current insertion position and advances the insertion position to the end of the string.

XwEditResult XwTextReplace(w, startPos, endPos, text)
XwTextEditWidget w;
XwTextPosition startPos,
endPos;
unsigned char *text;

Remove text in the source from startPos to endPos and insert the string text starting at startPos. If startPos and endPos are the same the action is an insertion. If text is the empty string, the action is a deletion.

Drawing and Updating

XwTextRedraw(w);
XwTextEditWidget w;

Refresh the widget screen.

void XwTextUpdate(w, status)
XwTextEditWidget w;
Boolean status;

This function turns the widget's screen updating function on and off. Wrapping these calls around a sequence of calls that change the content of the internal buffer will prevent screen flash.

End of Buffer

XwTextPosition XwTextGetLastPos (w, lastPos)
XwTextEditWidget w;
XwTextPosition lastPos;
This function returns the last character position in the buffer.

Insertion Position

void XwTextSetInsertPos(w, position)
XwTextEditWidget w;
XwTextPosition position;

XwTextPosition XwTextGetInsertPos(w)
XwTextEditWidget w;
These functions set and return the insertion position.

Setting the Source

void XwTextSetSource(w, source, startpos)
XwTextEditWidget w;
XwTextSourcePtr source;
XwTextPosition startpos;

SOURCE DEFINITION

The source provides textual data space and functions for manipulating that data. The functions are defined below. An application can define its own source by reimplementing these functions.

Read

XwTextPosition SourceRead(src, pos, text, maxread)
XwTextsource *src;
XwTextPosition pos;
XwTextblock *text;
XwTextPosition maxread;
This function returns a read-only text block in the src with maxread number of characters starting from pos. The return value is the next character position following the block.

Replace

XwEditResult SourceReplace(src, startpos, endpos, textblk, delta)
XwTextsource *src;
XwTextPosition startpos,
endpos;
XwTextBlock *textblk;
XwTextPosition *delta;
This function removes existing text in src between startpos and endpos and inserts new text from textblk at startpos. delta is change in the size of the text source. It returns XweditDone for a successful operation, XweditPosError for positional errors when source is in XttextAppend mode, and XweditError when the operation could not be performed.

SetLastPosition

XwTextPosition SourceSetLastPos(src, lastpos)
XwTextSource *src;
XwTextPosition lastpos;
This functions sets the last position in the source.

Scan

XwTextPosition SourceScan(src, pos, scantype, dir, count, include)
XwTextsource *src;
XwTextPosition pos;
XwScanType scantype;
XwScanDirection dir;
int count;
Boolean include;

SourceScan searches in dir direction (XwsdLeft XwsdRight) for XwScantype (XwstPositions, XwstWhiteSpace, XwstEOL, XwstParagraph, XwstLast). count is the number of the given type it will scan over and include indicates whether to count the item currently pointing at. It returns the starting position of the item scanned for.

EditType

XtEdittype SourceEditType(src)
XwTextsource *src;

Returns the edit type of source.

CURRENT LIMITATIONS

The current default source is not optimized for large amounts of data. X11's current selection is not yet supported.

ORIGIN

Digital Equipment Corporation. Massachusetts Institute of Technology. Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X)

NAME

XwtitlebarWidgetClass – An X Widget for creating titlebars.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/TitleBar.h>
```

CLASSES

A subclass of Core, Composite, Constraint and XwManagerClass.

The widget class to use when creating a TitleBar widget is **XwtitlebarWidgetClass**.

The class name of TitleBar is **TitleBar**.

DESCRIPTION

TitleBar provides a flexible mechanism for creating titlebars containing text and arbitrary widgets. Inputs are an optional text string and any number of widgets to manage. The title string will be displayed in a StaticText widget (refer to XWSTATICTEXT(3X)). Managed widgets may have optionally specified layout information (see CONSTRAINT RESOURCES below).

When TitleBar is directed to become narrower than is neccessary to display all of its interior widgets, some widgets may be hidden. The XtNprecedence resource in each managed widget controls this feature.

As TitleBar is directed to become narrower and narrower, widgets whose sum of XtNrPadding and XtNlPadding is greater than zero will have their padding collapsed to one pixel. Widgets will have their padding stripped in order of decreasing values of XtNprecedence.

If, after collapsing all of the widgets' padding, TitleBar is still too narrow to display all of its children widgets, widgets will be hidden. Widgets will be hidden in order of decreasing values of XtNprecedence. TitleBar will try to always display a widget of the highest priority (lowest value of XtNprecedence, even if it must be clipped.

Users of TitleBar should note that when children widgets are hidden they are completely hidden. Additionally, users who wish to make extensive use of the obscurability rules should read carefully the section on XtNprececence in the CONSTRAINT RESOURCES section below.

NEW RESOURCES

To specify any of these resources within a resource defaults file, simply drop the XtN prefix from the resource name. TitleBar defines the following new resources:

TitleBar Resource Set			
Name	**Class**	**Type**	**Default**
XtNtitlePrecedence	XtCTitlePrecedence	int	0
XtNtitleRegion	XtCTitleRegion	XwAlignment	XwALIGN_CENTER
XtNtitlePosition	XtCTitlePosition	int	0
XtNtitleRPadding	XtCTitleRPadding	int	1
XtNtitleLPadding	XtCTitleLPadding	int	1
XtNtitleForeground	XtCForeground	Pixel	black
XtNtitleBackground	XtCBackground	Pixel	white
XtNtitleHSpace	XtNTitleHSpace	int	2
XtNtitleVSpace	XtVTitleHSpace	int	2
XtNtitleBorderWidth	XtCBorderWidth	int	0
XtNtitleSelect	XtCCallback	Pointer	NULL
XtNtitleRelease	XtCCallback	Pointer	NULL
XtNhSpace	XtCHSpace	int	2
XtNvSpace	XtCVSpace	int	2
XtNenter	XtCCallback	Pointer	NULL
XtNleave	XtCCallback	Pointer	NULL
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

XtNtitlePrecedence
The value to be loaded into the constraint record of the optional StaticText widget.

XtNtitleRegion
The value to be loaded into the XtNtitleRegion constraint resource of the optional StaticText widget.

XtNtitlePosition
The value to be loaded into the XtNtitlePosition constraint resource of the optional StaticText widget.

XtNtitleRPadding
The value to be loaded into the XtNtitleRPadding constraint resource of the optional StaticText widget.

XtNtitleLPadding
The value to be loaded into the XtNtitleLPadding constraint resource of the optional StaticText widget.

XtNtitleForeground
The value to be loaded into the XtNforeground resource of the optional StaticText widget's core part.

XtNtitleBackground
The value to be loaded into the XtNbackground resource of the optional StaticText widget's core part.

XtNtitleHSpace
The value to be loaded into the XtNhSpace resource of the optional StaticText widget.

XtNtitleVSpace
The value to be loaded into the XtNvSpace resource of the optional StaticText widget.

XtNtitleBorderWidth
The value to loaded into the XtNborderWidth resource of the optional StaticText widget.

XtNtitleSelect
The value loaded into the XtNselect resource of the optional StaticText widget.

XtNtitleRelease
The value loaded into the XtNrelease resource of the optional StaticText widget.

XtNhSpace
The amount of space to maintain between the right and left of the titlebar and the interior widgets.

XtNvSpace
The amount of space to maintain between the top and bottom of the titlebar and the interior widgets.

XtNenter, XtNleave, XtNselect, and XtNrelease
Callbacks provided for control of TitleBar. The data parameter is unused.

INCORPORATED RESOURCES

The TitleBar creates an internal StaticText widget to handle the title string. In order to provide the user some control over the appearance of this internal widget, the following resources defined by StaticText are incorporated into TitleBar's resource list.

It must be noted that only the resources within the following tables will have any effect on the internal StaticText widget. The other resources defined for StaticText will be overridden by TitleBar.

For a complete description of the following resources, refer to the manual page given in the table heading.

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightStyle	XtCHighlightStyle	int	XwPATTERN_BORDER
XtNhighlightTile	XtCHighlightTile	int	XwBACKGROUND
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNtraversalType	XtCTraversalType	int	HIGHLIGHT_OFF

Static Text Resource Set -- STATICTEXT(3X)			
Name	Class	Type	Default
XtNstring	XtCString	char *	NULL
XtNalignment	XtCAlignment	XwAlignment	XwALIGN_CENTER
XtNwrap	XwCWrap	Boolean	TRUE
XtNlineSpace	XtCLineSpace	int	0
XtNfont	XtCFont	XFontStruct *	Fixed

CONSTRAINT RESOURCES

The following resources will be attached to every widget inserted into TitleBar. See CONSTRAINT(3X) for a general discussion of constraint resources.

TitleBar uses the constraint resources as hints during the layout of its managed children. Under certain conditions, any of these resources except XtNprecedence can (and will) be ignored by TitleBar.

Constraint Resource Set -- Children of TITLEBAR(3X)			
Name	Class	Type	Default
XtNregion	XtCRegion	XwAlignment	See below.
XtNposition	XtCPosition	int	0
XtNlPadding	XtCLPadding	int	2
XtNrPadding	XtCRPadding	int	2
XtNprecedence	XtCPrecedence	int	1

XtNregion

Associates a child with a region of the titlebar. The regions may be specified in the resource default file as "left" for XwALIGN_LEFT, "center" for XwALIGN_CENTER, and "right" for XwALIGN_RIGHT.

During layout widgets with XtNregion values of XwALIGN_LEFT grouped to the left end of TitleBar. Widgets with XtNregion values of XwALIGN_LEFT are grouped to the right of TitleBar. Widgets with XtNregion values of XwALIGN_CENTER will be grouped between the left and right groups. Additionally, TitleBar tries to center the center group within the TitleBar.

Widgets for which XtNregion is unspecified or XwALIGN_NONE when XtNstring is non-null, will be assigned one of the two regions not equal to XtNtitleRegion in an alternating fashion.

Widgets for which XtNregion is unspecified or XwALIGN_NONE when XtNstring is null, will be assigned a region. The first such widget will be assigned to the left region, the next to the center region, the next to the right region, the next to the left region, and so forth.

XtNposition

This resource gives the order of widgets within region. The left and the center region are layed out with XtNposition values increasing from left to right. The right region is laid out with XtNposition values increasing from right to left.

Position values are unique within a region. If two widgets are assigned the same position, the widget which was assigned first gets the position. The second widget gets the next available position. For example, widget1 and

widget2 are the only widgets inserted in TitleBar. Widget1 is inserted befor widget2. Widget1 and widget2 are both assigned a position of 4. Widget1 will be given the position of 4, and widget2 will be assigned a position of 5.

XtNlPadding

The number of pixels that TitleBar should try to maintain between the left of the widget and the right padding of the sibling widget to the left. For example, widget1 is to the left of widget2 within TitleBar. Widget1 has a XtNrPadding value of 5. Widget2 has a XtNlPadding value of 5. The borders of widget1 and widget2 will be 10 pixels apart.

If TitleBar is too narrow to honor all of its children's padding requests without hiding some children, some, possibly all, padding requests will be collapsed.

XtNrPadding

The number of pixels that TitleBar should try to maintain between the right of the widget and the left padding of the sibling widget to the right. For example, widget1 is to the right of widget2 within TitleBar. Widget1 has a XtNlPadding value of 5. Widget2 has a XtNrPadding value of 5. The borders of widget1 and widget2 will be 10 pixels apart.

If TitleBar is too narrow to honor all of its children's padding requests without hiding some children, some, possibly all, padding requests will be collapsed.

XtNprecedence

When TitleBar is too narrow to display all of its children, this resource is used to determine which children should be hidden. Widgets with high values of XtNprecedence are hidden first. Precedence values are relative to all other widgets within an instantiation of TitleBar. This means that all widgets, regardless of their region, with high values of XtNprecedence will be hidden before any widgets with the next lower values are hidden.

Values of XtNprecedence need not be unique. If values are unique, there is no question about which widget is first to lose its padding, nor about which widget is first to be hidden.

If values are not unique for all children of TitleBar, there need be no question about which widget is acted on first, but it is dependent on both insertion order and precedence. The last widget inserted in TitleBar of a given precedence is the first to lose its requested padding (of widgets with that priority). Widgets lose padding from last inserted to first inserted, within a given level of precedence. When hiding widgets, widgets within a given precedence level are hidden from last inserted to first inserted.

INHERITED RESOURCES

The following resources are inherited from the indicated superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

The input to the toggle is driven by the mouse buttons. The default translation set defining this button is listed below.

<EnterWindow>: enter()"
<LeaveWindow>: leave()"
<Btn1Down>: select()"
<Btn1Up>: release()"

ACTIONS

enter
If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE) and the parent of this widget is not a subclass of XwManager, initiate keyboard traversal. After this, the callback list is invoked.

leave
If keyboard traversal is active (argument type XtNtraversalOn with argument value TRUE) and the parent of this widget is not a subclass of XwManager, terminate keyboard traversal. After this, the callback list is invoked.

select
Invokes the select callback list.

release
Invokes the release callback list.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), CONSTRAINT(3X), XWMANAGER(3X), XWSTATICTEXT(3X), XWCREATETILE(3X)

NAME

XwtoggleWidgetClass – the X Widgets toggle button widget

SYNOPSIS

#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Toggle.h>

CLASSES

The toggle widget is built from the Core, XwPrimitive and XwButton classes.

The widget class to use when creating a toggle is **XwtoggleWidgetClass**. The class name is **Toggle**.

DESCRIPTION

The toggle widget implements a button which consists of a graphic and a label. The label can be positioned either to the right (the default) or the left of the graphic. The size of the graphic is based on the height of the font used for the label. The space between the graphic and the label is equal to 1/3 the font height. The default graphic is a square box and this may be changed to a diamond shape. It is intended that application writers can put a group of square buttons into a Row Column manager with its mode set to the default n_of_many to get the checkbox, or N of Many, selection semantic and then put a group of diamond buttons into a Row Column manager with its mode set to one_of_many to get the radiobutton, or One of Many, selection semantic.

The default semantic for this button is that button 1 down will toggle the state of the toggle. When in a selected state, the interior of the graphic will be filled with the foreground color; when not selected the interior of the graphic will be filled with the background color; when insensitive, the label will be drawn with the patterned tile (the default is a 75/25 mix of the foreground and background colors).

Callbacks can be attached to the widget to report selection (XtNselect) and unselection (XtNrelease). This widget can be set to respond to Enter and Leave window events by highlighting and unhighlighting the button. This widget is also capable of handling keyboard traversal. See the translations below for the default traversal keycodes.

NEW RESOURCES

The toggle widget class defines a set of resource types that can be used by the programmer to specify data for widgets of this class. Recall that the string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNfont becomes font.

Toggle Resource Set			
Name	**Class**	**Type**	**Default**
XtNsquare	XtCSquare	Boolean	True
XtNselectColor	XtCForeground	Pixel	Black

XtNsquare
If True, forces the button to draw a square box, otherwise it will draw a diamond shape box. One possible usage for this resource is to make the convention that row column managers containing diamond shaped toggles

have their XtNmode resource set to one_of_many which will only allow one of the buttons to be set at any one time, while row column managers containing square buttons use the default mode setting of n_of_many which allows any or all of the buttons to be set.

XtNselectColor
Allows the application to specify what color should be used to fill in the center of the square (or the diamond) when it is set.

INHERITED RESOURCES

The following resources are inherited from the named superclasses: The defaults used for the toggle when being created are as follows:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

Button Resource Set -- XWBUTTON(3X)			
Name	Class	Type	Default
XtNfont	XtCFont	XFontStruct *	Fixed
XtNlabel	XtCLabel	caddr_t	NULL
XtNlabelLocation	XtCLabelLocation	int	right
XtNvSpace	XtCVSpace	int	2
XtNhSpace	XtCHSpace	int	2
XtNset	XtCSet	Boolean	False
XtNsensitiveTile	XtCSensitiveTile	int	75 foreground

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to the toggle widget.

TRANSLATIONS

The input to the toggle is driven by the mouse buttons. The default translation set defining this button is listed below. Note that for the specific key symbols used in traversal, the HP Key Cap which corresponds to this key symbol appears to the right of the definition.

```
<Btn1Down>:      toggle()
<EnterWindow>:   enter()
<LeaveWindow>:   leave()
<Key>Select:     toggle()    HP "Select" key
```

ACTIONS

Note that this widget contains some actions which are not bound to any events by the default translations. The purpose of these additional actions are to allow advanced users to alter the button semantics to their liking.

toggle:

Toggle the set state of the button (make it TRUE if it was FALSE, FALSE if it was TRUE). Redraw only the toggle part (not the label) of the button. If the current state of the button is set (TRUE) issue the XtNselect callbacks, if not set (FALSE) issue the XtNrelease callbacks. No additional data beyond the widget id and the specified closure is sent with these callbacks.

enter:

If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the button will be highlighted. Otherwise no action is taken.

leave:

If the XtNtraversalType resouces has been set to XwHIGHLIGHT_ENTER then the button will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWBUTTON(3X)

NAME

XwvPanedWidgetClass – the X Widgets vertical paned manager widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/VPW.h>
```

CLASSES

The vertical paned manager widget is built out of the Core, Composite, Constraint and XwManager classes. Note that since the Composite class contains no user settable resources, there is no table for Composite class resources.

The widget class to use when creating a vertical paned manager is **XwvPanedWidgetClass.** The class name is **VPanedWindow.**

DESCRIPTION

The vertical paned manager is a composite widget which lays children out in a vertically tiled format. Children appear in a top to bottom fashion, with the first child inserted appearing at the top of the paned widget and the last child inserted appearing at the bottom of the paned widget. The vertical paned manager will grow to match the width of its widest child and all other children are forced to this width. The vertical paned mananger does not grow if setValues is performed on a child, making it the widest child. It is clipped instead. The height of the vertical paned manager will be equal to the sum of the heights of all its children and the (optional) padding surrounding them.

It is also possible for the end user to adjust the size of the panes. To facilitate this adjustment, a control widget (XwsashWidgetClass) is created for most children. The control widget appears as a square box positioned on the bottom of the pane which it controls. Using the mouse (see the description on translations below) a user can adjust the size of a pane.

The vertical paned manager is a constraint widget, which means that it creates and manages a set of constraints for each child. It is possible to specify a minimum and maximum size for each pane. The vertical paned widget will not allow a pane to be adjusted below its minimum nor beyond its maximum. Also, when the minimum size of a pane is equal to its maximum then no control widget will be presented for that pane. Nor will a control widget be presented for the bottom-most pane.

The vertical paned manager supports 2 presentation modes: framed and unframed. When framed, each pane is offset from the edges of the vertical paned manager and from other panes by a specified (and settable) number of pixels. In this mode the entire borderwidth of each child is also visible. Note that the vertical paned manager enforces a particular (and settable) border width on each pane. The second mode is unframed where the edge of a pane exactly corresponds to the edge of the vertical paned manager so that only a border between panes is visible.

No callbacks are defined for this manager.

NEW RESOURCES

The vertical paned manager defines a set of resource types used by the programmer to specify data for the manager widget. The programmer can also set the values for the Core and XwManager widget classes to set attributes for this widget. The following table contains the settable resources defined by the vertical paned manager. Recall that the string to be used when setting any of these resources in an application defaults file (like .Xdefaults) can be obtained by stripping the preface "XtN" off of the resource name. For instance, XtNmin becomes min.

Vertical Paned Resource Set			
Name	Class	Type	Default
XtNsashIndent	XtCSashIndent	int	-10
XtNborderFrame	XtCBorderWidth	int	1
XtNframed	XtCBoolean	Boolean	TRUE
XtNpadding	XtCPadding	int	3
XtNrefigureMode	XtCBoolean	Boolean	TRUE

XtNsashIndent

This controls where along the bottom of the pane the control widget (the pane's sash) will be placed. A positive number will cause the sash to be offset from the left side of the pane, a negative number will cause the sash to be offset from the right side of the pane. If the offset specified is greater than the width of the vertical paned manager, minus the width of the sash, the sash will be placed flush against the left hand side of the paned manager.

XtNborderFrame

The application can specify the thickness of the borderwidth of all panes in the paned manager. The value must be greater than or equal to 0.

XtNframed

The application can specify whether the panes should be displayed with some padding surrounding each pane (TRUE) or whether the panes should be set flush with the paned manager (FALSE).

XtNpadding

The application can specify how many pixels of padding should surround each pane when it is being displayed in framed mode. This value must be greater than or equal to 0.

XtNrefigureMode

This setting is useful if a large number of programmatic manipulations are taking place. It will prevent the manager from recomputing and displaying new positions for the child panes (FALSE). Once the changes have been executed this flag should be set to TRUE to allow the vertical paned manager to show the correct positions of the current children.

CONSTRAINT RESOURCES

The following resources are attached to every widget inserted into vertical paned manager. See *CONSTRAINT(3X)* for a general discussion of constraint resources.

Constraint Resource Set -- Children of VPANEDWINDOW(3X)			
Name	**Class**	**Type**	**Default**
XtNmin	XtCMin	int	1
XtNmax	XtCMax	int	10000
XtNallowResize	XtCBoolean	Boolean	FALSE
XtNskipAdjust	XtCBoolean	Boolean	FALSE

XtNmin
Allows an application to specify the mimimum size to which a pane may be resized. This value must be greater than 0.

XtNmax
Allows an application to specify the maximum size to which a pane may be resized. This value must be greater than the specified minimum.

XtNallowResize
Allows an application to specify whether the vertical paned manager should allow a pane to request to be resized. This flag only has an effect after the paned manager and its children have been realized. If this flag is set to TRUE, the manager will try to honor requests to alter the height of the pane. If false, it will always deny pane requests to resize.

XtNskipAdjust
Allows an application to specify that the vertical paned manager should not automatically resize this pane (flag set to TRUE).

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Manager Resource Set -- XWMANAGER(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalOn	XtCTraversalOn	Boolean	FALSE
XtNlayout	XtCLayout	int	minimize
XtNnextTop	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalOn resource is set to TRUE at either create time or during a call to XtSetValues, the XwManager superclass will automatically augment the manager widget's translations to support keyboard traversal. Refer to the XwManager man page for a complete description of these translations.

SASH TRANSLATIONS

The translations which control the sashes created for each adjustable pane are replicated here for convenience.

```
<Btn1Down>:      SashAction(Start, UpperPane)
<Btn2Down>:      SashAction(Start, ThisBorderOnly)
<Btn3Down>:      SashAction(Start, LowerPane)
<Btn1Motion>:    SashAction(Move, Upper)
<Btn2Motion>:    SashAction(Move, ThisBorder)
<Btn3Motion>:    SashAction(Move, Lower)
Any<BtnUp>:      SashAction(Commit)
<EnterWindow>:   enter()
<LeaveWindow>:   leave()
```

SashAction(Start, UpperPane):
Change the cursor from the crosshair to an upward pointing arrow. Determine the upper pane which will be adjusted (usually the pane to which the sash is attached).

SashAction(Start, ThisBorderOnly):
Change the cursor from the crosshair to a double headed arrow. The panes that will be adjusted are the pane to which the sash is attached and the first pane below it that can be adjusted. Unlike the UpperPane and LowerPane mode, only 2 panes will be effected. If one of the panes reaches its minimum or maximum, adjustment will stop, instead of finding the next adjustable pane.

SashAction(Start, LowerPane):
Change the cursor from the crosshair to a downward pointing arrow. Determine the lower pane which will be adjusted (usually the pane below the pane to which the sash is attached).

SashAction(Move, Upper):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget below the upper pane can be adjusted and make the appropriate adjustments.

SashAction(Move, ThisBorder):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Adjust as needed (and as possible) the upper and lower panes selected when the SashAction(Start, ThisBorderOnly) action was invoked.

SashAction(Move, Lower):
Draw a series of track lines to illustrate what the heights of the panes would be if the Commit action were invoked. Determine which widget above the lower pane can be adjusted and make the appropriate adjustments.

enter:
Enter window events occurring on the scrolled window are handled by this action.

leave:
Leave window events occurring on the scrolled window are handled by this action.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWMANAGER(3X), XWPRIMITIVE(3X), XWSASH(3X)

NAME

XwvaluatorWidgetClass – the X Widget's valuator widget

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/Valuator.h>
```

CLASSES

The Valuator widget is built from the Core and XwPrimitive classes.

The widget class to use when creating a valuator is **XwvaluatorWidgetClass**. The class name for Valuator is **Valuator**.

DESCRIPTION

The Valuator widget implements a horizontal or vertical scrolling widget as a rectangular bar containing a sliding box (slider). The Valuator widget supports input through interactive slider movement and selections on the slide area not occupied by the slider. Both types of input have a separate callback list for communicating with the application. The Valuator widget can be used by the application to attach to objects scrolled under application control, or used by composite widgets to implement predefined scrolled objects.

NEW RESOURCES

The Valuator widget defines a set of resource types used by the programmer to specify the data for the valuator. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget. To reference a resource in a .Xdefaults file, strip off the XtN from the resource string. The following table contains the set of resources defined by Valuator.

Valuator Resource Set			
Name	**Class**	**Type**	**Default**
XtNsliderMin	XtCSliderMin	int	0
XtNsliderMax	XtCSliderMax	int	100
XtNsliderExtent	XtCSliderExtent	int	10
XtNsliderOrigin	XtCSliderOrigin	int	0
XtNsliderTile	XtCSliderTile	int	foreground
XtNslideOrientation	XtCSlideOrientation	int	vertical
XtNsliderMoved	XtCCallback	Pointer	NULL
XtNsliderReleased	XtCCallback	Pointer	NULL
XtNareaSelected	XtCCallback	Pointer	NULL

XtNsliderMin, XtNsliderMax
The Valuator widget lets the application define its own coordinate system for the valuator. Any integer values with sliderMin less than sliderMax can be specified.

XtNsliderExtent
The size of the slider can be set by the application. The acceptable values are 0 < sliderExtent < (sliderMax - sliderMin).

XtNsliderOrigin
The location of the slider can be set by the application. The acceptable values are between sliderMin and (sliderMax - sliderExtent).

XtNsliderTile
This resource is used to set the tile used to create the pixmap to use when drawing the slider. The #defines for setting the values through an arg list and the strings to be used in the .Xdefaults files are described in XwCreateTile(3X).

XtNslideOrientation
The Valuator widget supports both horizontal and vertical scrolling. This resource type is the means by which this is set. It can be defined through the .Xdefaults file by the strings "horizontal", and "vertical" or within an arg list for use in XtSetValues() by the defines XwHORIZONTAL and XwVERTICAL.

XtNsliderMoved, XtNsliderReleased, XtNareaSelected
The Valuator widget defines three types of callback lists which get invoked upon different event conditions when interacting with a valuator. All types use the data parameter to send the location of the slider to the callback functions.

The first callback type, sliderMoved, defines the callback list containing the callback functions called when the slider is interactively moved.

The second callback type, sliderReleased, defines a callback list containing callback functions called when the mouse button is released after moving the slider.

The third callback type, areaSelected, defines a callback list containing the callback functions called when an area in a valuator not containing the slider is selected. The slider is not moved to this position but if the application wants the slider moved, it can use the position value contained in the parameter call_data and perform a XtSetValues() to its valuator.

For the callback types, the call_data parameter of the callback function will be an integer containing the slider or selection position.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	Class	Type	Default
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	Class	Type	Default
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at either create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. See the XwPrimitive man page for a complete description of these translations. See the TRANSLATIONS section in this man page for a description of the translations local this widget.

TRANSLATIONS

The input to the Valuator widget is driven by the mouse buttons. The default translation is defined as follows:

```
<Btn1Down>:          select(),
<Btn1Up>:            release(),
Button1<PtrMoved>:   moved(),
<EnterWindow>:       enter(),
<LeaveWindow>:       leave(),
Ctrl<Key>Left:       left(),     HP "Control Left Cursor" key
Ctrl<Key>Up:         up(),       HP "Control Up Cursor" key
Ctrl<Key>Right:      right(),    HP "Control Right Cursor" key
Ctrl<Key>Down:       down(),     HP "Control Down Cursor" key
```

ACTIONS

select:
Select processes the activation conditions within the valuator, both for selections within the slider area and on the slider.

release:
Release handles the processing terminating conditions for selections on the valuator.

moved:
Moved processes interactive movement of the slider following a selection upon the slider.

enter:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be highlighted. Otherwise no action is taken.

leave:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the arrow's border will be unhighlighted. Otherwise no action is taken.

left: If the valuator's orientation is horizontal, this action will cause its slider origin to be decremented by 1 unit and redisplayed.

up: If the valuator's orientation is vertical, this action will cause its slider origin to be decremented by 1 unit and redisplayed.

right:
If the valuator's orientation is horizontal, this action will cause its slider origin to be incremented by 1 unit and redisplayed.

down:
If the valuator's orientation is vertical, this action will cause its slider origin to be incremented by 1 unit and redisplayed.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X), XWCREATETILE(3X)

NAME

XwworkSpaceWidgetClass – the X Widget's empty window widget.

SYNOPSIS

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xw/Xw.h>
#include <Xw/WorkSpace.h>
```

CLASSES

The WorkSpace widget is built from the Core and XwPrimitive classes.

The widget class to use when creating a workspace is **XwworkSpaceWidgetClass.**

The class name for this widget is **WorkSpace.**

DESCRIPTION

The WorkSpace widget provides the application developer with an empty primitive widget. This widget can be used by the application as a non-widget graphics area. Callback types are defined for widget exposure and resize to allow the application to redraw or reposition its graphics. Keyboard, button press and button release callbacks are also defined to provide the application an easy means of getting normal input from the widget. Other types of input can be gathered from the widget by adding event handlers.

If the workspace widget has a highlight thickness, the application should take care not to draw on this area. This can be done by creating the graphics context to be used for drawing in the widget with a clipping rectangle set to the size of the widget's window inset by the highlight thickness.

NEW RESOURCES

The WorkSpace widget defines a set of resource types used by the programmer to specify the data for the workspace. The programmer can also set the values for the Core and Primitive widget classes to set attributes for this widget.

WorkSpace Resource Set			
Name	**Class**	**Type**	**Default**
XtNexpose	XtCCallback	Pointer	Null
XtNresize	XtCCallback	Pointer	Null
XtNkeyDown	XtCCallback	Pointer	Null

XtNexpose
This resource defines a callback list which is invoked when an exposure event occurs on the widget. The call_data parameter for the callback will contain a Region structure containing the exposed region.

XtNresize
This resource defines a callback list which is invoked when the widget is resized. The widget parameter can be accessed to obtain the new size of the widget.

XtNkeyDown
This resource defines a callback list which is invoked when keyboard input occurs in the widget. The call_data parameter for the callback will contain the key pressed event.

INHERITED RESOURCES

The following resources are inherited from the named superclasses:

Core Resource Set -- CORE(3X)			
Name	**Class**	**Type**	**Default**
XtNancestorSensitive	XtCSenstitive	Boolean	TRUE
XtNx	XtCPosition	int	0
XtNy	XtCPosition	int	0
XtNwidth	XtCWidth	int	0
XtNheight	XtCHeight	int	0
XtNdepth	XtCDepth	int	0
XtNbackground	XtCBackground	Pixel	White
XtNbackgroundPixmap	XtCPixmap	Pixmap	Unspecified
XtNborderWidth	XtCBorderWidth	int	1
XtNborderColor	XtCBorderColor	Pixel	Black
XtNborderPixmap	XtCPixmap	Pixmap	Unspecified
XtNsensitive	XtCSensitive	Boolean	TRUE
XtNmappedWhenManaged	XtCMappedWhenManaged	Boolean	TRUE
XtNdestroyCallback	XtCCallback	Pointer	NULL
XtNtranslations	XtCTranslations	XtTranslations	NULL

Primitive Resource Set -- XWPRIMITIVE(3X)			
Name	**Class**	**Type**	**Default**
XtNforeground	XtCForeground	Pixel	Black
XtNbackgroundTile	XtCBackgroundTile	int	background
XtNtraversalType	XtCTraversalType	int	highlight_off
XtNhighlightStyle	XtCHighlightStyle	int	pattern_border
XtNhighlightColor	XtCForeground	Pixel	Black
XtNhighlightTile	XtCHighlightTile	int	50_foreground
XtNhighlightThickness	XtCHighlightThickness	int	0
XtNrecomputeSize	XtCRecomputeSize	Boolean	TRUE
XtNselect	XtCCallback	Pointer	NULL
XtNrelease	XtCCallback	Pointer	NULL

KEYBOARD TRAVERSAL

If the XtNtraversalType resource is set to highlight_traversal (XwHIGHLIGHT_TRAVERSAL in an argument list) at create time or during a call to XtSetValues, the XwPrimitive superclass will automatically augment the primitive widget's translations to support keyboard traversal. Refer to the XwPrimitive man page for a complete description of these translations. Refer to the TRANSLATIONS section in this man page for a description of the translations local to this widget.

TRANSLATIONS

The following translations are defined for the WorkSpace widget.

```
<KeyDown>:      keydown()
<BtnDown>:      select()
<BtnUp>:        release()
<EnterWindow>:  enter()
<LeaveWindow>:  leave()
```

ACTIONS

keydown:
Keyboard input occurring on a workspace invokes the workspace's XtNkeyDown callback list.

select:
Selections occurring on a workspace invokes the workspace's primitive XtNselect callback list.

release:
Release invokes the workspace's primitive XtNrelease callback list.

enter:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the workspace's border will be highlighted. Otherwise no action is taken.

leave:
If the XtNtraversalType resource has been set to XwHIGHLIGHT_ENTER then the workspace's border will be unhighlighted. Otherwise no action is taken.

ORIGIN

Hewlett-Packard Company.

SEE ALSO

CORE(3X), XWPRIMITIVE(3X)