ULTRIX
Worksystem Software

digital

Reference Pages,
Sections 3Dwt, 3X11, and 3Xt

Order Number: AA–MA99B–TE

ULTRIX Worksystem Software

Reference Pages, Sections 3Dwt, 3X11, and 3Xt

Order Number: AA-MA99B-TE

Product Version: ULTRIX Worksystem Software, Version 2.2
Operating System and Version: ULTRIX–32, Version 3.1 or higher

digital equipment corporation
maynard, massachusetts

This manual was written and produced by the Open Software Publications group.

About This Manual

Organization

The ULTRIX Worksystem Software *Reference Pages, Sections 3Dwt, 3X11, and 3Xt* contain the reference pages for the XUI Toolkit functions, the XUI Toolkit intrinsics, and the Xlib functions.

Format

Each reference page has the following general format.

Each has a title header consisting of the subject name and the appropriate section number, for example, `DwtAttachedDB`(3Dwt), `XOpenDisplay`(3X11), and `XtCreateApplicationContext`(3Xt).

The remaining subsections provide the specific information that is relevant to the topic. In general, the following subsection titles are used where appropriate:

Name

Lists the topic name and a short description of the entry.

Syntax

Provides the function definition. **Boldface** indicates characters typed literally. *Italics* indicates variable information that is to be specified by the user. An ellipsis (...) indicates that the preceding argument can be repeated. Square brackets [ ] enclose optional arguments.

Arguments

Describes each arguments that passed to or returned by the function.

Description

Describes the function, its usage, and its effects. Note that all references to chapters and sections in the Description section are for the respective, companion descriptive guide listed in the See Also section at the end of that page.

Diagnostics

Describes the diagnostic and error messages that may appear. In most cases, self-explanatory messages are not listed.

Restrictions
Describes all known restrictions or limitations for that function.

Files
Lists the related files that are either used or created by the function.

See Also
Lists references to related functions in the same library and to other, related documents.

Related Documents

XUI Style Guide

Describes the XUI user interface and, hence, the "look and feel" of an XUI application.

Guide to Writing Applications Using XUI Toolkit Widgets

Describes how to create an application using the XUI Toolkit.

Guide to the XUI Toolkit: C Language Binding

Describes the widgets (user interface abstractions) that you can use to write your XUI-based application.

Guide to the XUI Toolkit Intrinsics: C Language Binding

Describes the Intrinsics functions that you can use to write your XUI-based application or widget.

Guide to the Xlib Library: C Language Binding

Describes the low-level C functions that you can use to write your X-based application.

X Window System Protocol: X Version 11

Describes the precise semantics of the X11 protocol specification.

Conventions

The following typeface conventions are used in this manual:

`special`	In text, all function names, events, errors, constant names, and pathnames are presented in this type.
UPPERCASE	Although the ULTRIX system differentiates between lowercase and uppercase characters, uppercase is used intentionally in this manual where it is applicable.

In addition, the following conventions are used in this manual:

- To eliminate any ambiguity between those arguments that you pass and those that a function returns to you, the explanations for all arguments

that you pass start with the word *specifies* or, in the case of multiple arguments, the word *specify*. The explanations for all arguments that are returned to you start with the word *returns* or, in the case of multiple arguments, the word *return*. The explanations for all arguments that you can pass and are returned start with the words *specifies and returns*.

- Any pointer to a structure that is used to return a value is designated as such by the _return suffix as part of its name. All other pointers passed to these functions are used for reading only. A few arguments use pointers to structures that are used for both input and output and are indicated by the _in_out suffix.

Insert tabbed
divider here.
Then discard
this sheet.

Name

DwtActivateWidget – Allows the application to simulate push button activation.

Syntax

void DwtActivateWidget(*widget*)
Widget *widget*;

Arguments

widget	Specifies a pointer to the widget data structure.

Description

The `DwtActivateWidget` function allows the application to simulate push button activation. `DwtActivateWidget` generates the same highlighting and callbacks that would occur if the user clicks on a push button. For example, an application might contain functions that a user could choose either by selecting a menu option or by activating a push button. If the user selected the menu option, the application could activate the corresponding push button to maintain a consistent user interface. Only push buttons are currently supported by `DwtActivateWidget`.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtAddFontList(3Dwt)

Name

DwtAddFontList – Adds an entry to a font list.

Syntax

DwtFontList DwtAddFontList(*list*, *font*, *charset*)
DwtFontList *list*;
XFontStruct **font*;
long *charset*;

Arguments

list	Specifies a pointer to the font list to which an entry will be added.
font	Specifies a pointer to the font structure to be added to the list.
charset	Specifies the character set identifier for the font. Values for this argument can be found in the required file `/usr/include/cda_def.h`.

Description

The `DwtAddFontList` function adds an entry to a font list.

Return Value

This function returns the new font list.

See Also

DwtCreateFontList (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtAttachedDB, DwtAttachedDBCreate, DwtAttachedDBPopupCreate – Creates an attached dialog box or a pop-up attached dialog box widget to contain other information/request (dialog) subwidgets.

Syntax

Widget DwtAttachedDB(*parent_widget*, *name*, *default_position*,
x, *y*, *title*, *style*,
map_callback, *help_callback*)
Widget *parent_widget*;
char **name*;
Boolean *default_position*;
Position *x*, *y*;
DwtCompString *title*;
unsigned char *style*;
DwtCallbackPtr *map_callback*, *help_callback*;

Widget DwtAttachedDBCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Widget DwtAttachedDBPopupCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, causes `DwtNx` and `DwtNy` to be ignored and forces the default widget position. The default widget position is centered in the parent window. If `False`, the specified `DwtNx` and

DwtNy attributes are used to position the widget. This argument sets the DwtNdefaultPosition attribute associated with DwtDialogBoxCreate.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNx core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNy core widget attribute.

title Specifies the compound-string label. The label is given to the window manager for the title bar if the DwtNstyle attribute associated with DwtDialogBoxPopupCreate is DwtModal or DwtModeless. However, the label is used in the border if the DwtNstyle attribute associated with DwtDialogBoxCreate is DwtWorkarea.

The attribute name associated with this argument is DwtNtitle.

style Specifies the style of the dialog box widget. You can pass DwtModal, DwtModeless, or DwtWorkarea. This argument sets the DwtNstyle attribute associated with DwtDialogBoxCreate.

map_callback Specifies the callback function or functions called when the window is about to be mapped. For this callback, the reason is DwtCRMap. This argument is ignored if DwtNstyle is DwtWorkarea.

This argument sets the DwtNmapCallback attribute associated with DwtDialogBoxPopupCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtAttachedDB` and `DwtAttachedDBCreate` functions create an instance of an attached dialog box widget or an attached dialog box pop-up widget and return its associated widget ID. The `DwtAttachedDBPopupCreate` function creates an instance of a pop-up attached dialog box widget and returns its associated widget ID. The attached dialog box acts as a container only, and provides no input semantics over and above the semantics of the widgets that it contains. It differs from the dialog box in its handling of child widgets. Constraints are placed on each child widget at the time of creation. The default values for the constraint attributes are placed on the child unless you specify values for the constraint attributes. You specify these values either in the *override_arglist* or by calling `XtSetValues`.

By using the constraint attributes, you can attach each of the four sides of a child widget (top, bottom, right side, and left side) to a side of the parent attached dialog box, a side of another child widget, to a relative position within the attached dialog box, to itself, or to nothing. The possible attachments for each of the four sides are described in the Constraint Attributes section. Specifying these attachments allows you to maintain the position of child widgets within the attached dialog box as resizing occurs.

If only one attachment in a direction is specified with no width or height, the default width or height for the widget is used.

For all attachment types, you can optionally specify an offset in pixels or font units. The offset determines the amount of space between the side of the child widget and the side or position you attach it to. By default, the child widgets are positioned in an attached dialog box in terms of font units rather than pixel units. (That is, `DwtNunits` is `DwtFontUnits`.) The X font units are defined to be one-fourth the width of whatever font is supplied for the common attribute `DwtNfont`. The Y font units are defined to be one-eighth the width of whatever font is supplied for `DwtNfont`.

The offsets given are automatically negated when dealing with right and bottom sides. For example, a displacement of 5 means that the side stays 5 units to the right of its attachment if a left side, and 5 units to the left if a right side.

Displacements default to a value specified in the attached dialog box for attachments to the attached dialog box and the widget, and half the value specified if attached to a position. Attaching to a point allows several widgets to grow proportionally; the space between them should be the default displacement. There are separate horizontal and vertical defaults.

DwtAttachedDB (3Dwt)

You can determine whether the attached dialog box will honor resize geometry requests from a given child widget by appropriately setting the DwtNresize attribute for that child. If it does honor a request, the attached dialog box reconfigures all child widgets based on the initial coordinate information. You can add child widgets after the attached dialog box widget has been realized. If there is extra room in the attached dialog box, the new child widget will appear. If there is not enough room, the attached dialog box will ask the geometry manager for permission to resize.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometr manager
DwtNy	Position	Determined by the geometr manager
DwtNwidth	Dimension	Widget-specific
DwtNheight	Dimension	Widget-specific
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensit attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent windo
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL
Constraint Attributes		

DwtNadbTopAttachment	DwtAttachmentType	DwtAttachAdb if DwtNrubberPositioning is False DwtAttachSelf if DwtNrubberPositioning is True
DwtNadbBottomAttachment	DwtAttachmentType	The default is DwtAttachNone if DwtNrubberPositioning is False. The default is DwtAttachSelf if DwtNrubberPositioning is True.
DwtNadbLeftAttachment	DwtAttachmentType	The default is DwtAttachAdb if DwtNrubberPositioning is False. The default is DwtAttachSelf if DwtNrubberPositioning is True.
DwtNadbRightAttachment	DwtAttachmentType	The default is DwtAttachNone if DwtNrubberPositioning is False. The default is DwtAttachSelf if DwtNrubberPositioning is True.
DwtNadbTopWidget	Widget	NULL
DwtNadbBottomWidget	Widget	NULL
DwtNadbLeftWidget	Widget	NULL
DwtNadbRightWidget	Widget	NULL
DwtNadbTopPosition	int	Zero
DwtNadbBottomPosition	int	Zero
DwtNadbLeftPosition	int	Zero
DwtNadbRightPosition	int	Zero
DwtNadbTopOffset	int	The value specified with DwtNdefaultVerticalOffset. However, if DwtNadbTopAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value specified with DwtNdefaultVerticalOffset.

DwtNadbBottomOffset	int	The default is the value specified with DwtNdefaultVerticalOffset. However, if DwtNadbBottomAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value specified with DwtNdefaultVerticalOffset.
DwtNadbLeftOffset	int	The default is the value specified with DwtNdefaultHorizontalOffs However, if DwtNadbLeftAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value of DwtNdefaultHorizontalOffs
DwtNadbRightOffset	int	The value specified with DwtNdefaultHorizontalOffs However, if DwtNadbRightAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value specified with DwtNdefaultHorizontalOffs
DwtNresizable	Boolean	True

Dialog Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNunits	unsigned char	DwtFontUnits

DwtNstyle	unsigned char	For DwtDialogBoxCreate, the default is DwtWorkarea. For DwtDialogBoxPopupCreate, the default is DwtModeless.
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	XtTranslations	NULL
DwtNmarginWidth	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreate, the default is 3 pixels.
DwtNmarginHeight	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreate, the default is 3 pixels.
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeGrowOnly
DwtNgrabKeySyms	KeySym	The default array contains the Tab key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT()\n\ Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()";

The following constraint attributes belong to any widget that is made a child of an attached dialog box widget. You cannot set these attributes on the attached dialog box itself; you must set them on the child widget. Several of these constraint attributes take an enumerated data type. You should not change attachment attributes in an attached dialog box with XtSetValues, as this could result in an infinite loop.

```
typedef enum _DwtAttachmentType {
                        DwtAttachNone,
                        DwtAttachAdb,
                        DwtAttachWidget,
                        DwtAttachPosition,
                        DwtAttachSelf,
                        DwtAttachOppWidget,
                        DwtAttachOppAdb,
```

```
} DwtAttachmentType;
```

`DwtNadbTopAttachment`

Specifies how the top side of the child widget is attached to its parent attached dialog box widget, another child widget, a position, or itself.

The following describes the enumerated data type values as they apply to this attribute:

Value	Meaning
`DwtAttachNone`	Do not attach this side. This type of attachment may be overridden by the defaults of other attachments that affect this side.
`DwtAttachAdb`	Attach the top side of the child widget to the top side of its parent attached dialog box.
`DwtAttachOppAdb`	Attach the top side of the child widget to the bottom side of its parent attached dialog box.
`DwtAttachWidget`	Attach the top side of the child widget to the bottom side of another child widget within the parent attached dialog box.
`DwtAttachOppWidget`	Attach the top side of the child widget to the top side of another child widget.
`DwtAttachPosition`	Attach the top side of the child widget to a relative position inside the parent attached dialog box. Specify the relative position as a fraction of the total width or height of the attached dialog box.
`DwtAttachSelf`	Attach the top side of the child widget to a relative position corresponding to the side's initial position in the attached dialog box.

`DwtNadbBottomAttachment`

Specifies how the bottom side of the widget is attached to the side of its parent attached dialog box widget, another child widget, a position, or itself.

The following describes the enumerated data type values as they apply to this attribute:

Value	Meaning
DwtAttachNone	Do not attach this side. This type of attachment overrides any default attachment that might affect the side.
DwtAttachAdb	Attach this side to the bottom side of its parent attached dialog box.
DwtAttachOppAdb	Attach this side to the top side of the parent attached dialog box.
DwtAttachWidget	Attach this side to the top side of another child widget within the parent attached dialog box.
DwtAttachOppWidget	Attach this side to the bottom side of another child widget.
DwtAttachPosition	Attach this side to a relative position inside the parent attached dialog box. Specify the relative position as a fraction of the total width or height of the attached dialog box.
DwtAttachSelf	Attach this to a relative position corresponding to the side's initial position inside the parent attached dialog box.

DwtNadbLeftAttachment

Specifies how the left side of the widget is attached to the side of its parent attached dialog box widget, another child widget, a position, or itself.

The following describes the enumerated data type values as they apply to this attribute:

Value	Meaning
DwtAttachNone	Do not attach this side. This type of attachment overrides any default attachment that might affect the side.
DwtAttachAdb	Attach this side to the left side of its parent attached dialog box.
DwtAttachOppAdb	Attach this side to the right side of the parent attached dialog box.

DwtAttachWidget	Attach this side to the right side of another child widget within the parent attached dialog box.
DwtAttachOppWidget	Attach this side to the left side of another child widget.
DwtAttachPosition	Attach this side to a relative position inside the parent attached dialog box. Specify the relative position as a fraction of the total width or height of the attached dialog box.
DwtAttachSelf	Attach this side to a relative position corresponding to the side's initial position in the parent attached dialog box.

DwtNadbRightAttachment

Specifies how the right side of the widget is attached to the side of its parent attached dialog box, another child widget, a position, or itself.

The following describes the enumerated data type values as they apply to this attribute:

Value	Meaning
DwtAttachNone	Do not attach this side. This type of attachment overrides any default attachment that might affect the side.
DwtAttachAdb	Attach this side to the right side of its parent attached dialog box.
DwtAttachOppAdb	Attach this side to the left side of the parent attached dialog box.
DwtAttachWidget	Attach this side to the left side of another child widget within the parent attached dialog box.
DwtAttachOppWidget	Attach this side to the right side of another child widget.
DwtAttachPosition	Attach this side to a relative position inside the parent attached dialog box. Specify the relative position as a fraction of the total width or height of the attached dialog box.

DwtAttachSelf	Attach this side to a relative position corresponding to the side's initial position in the parent attached dialog box.

DwtNadbTopWidget
: Specifies the child widget that the top side is attached to if DwtNadbTopAttachment is DwtAttachWidget or DwtAttachOppWidget. Otherwise, this attribute is ignored.

DwtNadbBottomWidget
: Specifies the widget that the bottom side is attached to if DwtNadbBottomAttachment is DwtAttachWidget or DwtAttachOppWidget. Otherwise, this attribute is ignored.

DwtNadbLeftWidget
: Specifies the widget that the left side is attached to if DwtNadbLeftAttachment is DwtAttachWidget or DwtAttachOppWidget. Otherwise, this attribute is ignored.

DwtNadbRightWidget
: Specifies the widget that the right side is attached to if DwtNadbRightAttachment is DwtAttachWidget or DwtAttachOppWidget. Otherwise, this attribute is ignored.

DwtNtopPosition
: Specifies the numerator used with DwtNfractionBase to determine the relative positioning of the top side if DwtNadbTopAttachment is DwtAttachPosition. Otherwise, this attribute is ignored.

DwtNadbBottomPosition
: Specifies the numerator used with DwtNfractionBase to determine the relative positioning of the bottom side if DwtNadbBottomAttachment is DwtAttachPosition. Otherwise, this attribute is ignored.

DwtNadbLeftPosition
: Specifies the numerator used with DwtNfractionBase to determine the relative

positioning of the left side if DwtNadbLeftAttachment is DwtAttachPosition. Otherwise, this attribute is ignored.

DwtNadbRightPosition
Specifies the numerator used with the DwtNfractionBase to determine the relative positioning of the right side if DwtNadbRightAttachment is DwtAttachPosition. Otherwise, this attribute is ignored.

DwtNadbTopOffset
Specifies the offset of the top side from the position, widget, or attached dialog box.

DwtNadbBottomOffset
Specifies the offset of the bottom side from the position, widget, or attached dialog box.

DwtNadbLeftOffset
Specifies the offset of the left side from the position, widget, or attached dialog box.

DwtNadbRightOffset
Specifies the offset of the right side from the position, widget, or attached dialog box.

DwtNresizable — Specifies a boolean value that, when True, indicates that the attached dialog box can change the size of the child widget. If False, indicates that the attached dialog box cannot change the size of the child widget.

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNdefaultHorizontalOffset	int	Zero
DwtNdefaultVerticalOffset	int	Zero
DwtNrubberPositioning	Boolean	False
DwtNfractionBase	int	100

DwtNdefaultHorizontalOffset
: Specifies the default horizontal offset for right and left attachments. The offset determines the amount of space between the left or right side of a child widget and the side or position to which it is attached.

DwtNdefaultVerticalOffset
: Specifies the default vertical offset for the top and bottom attachments. The offset determines the amount of space between the top or bottom side of a child widget and the side or position to which it is attached.

DwtNrubberPositioning
: Specifies a boolean value that, when `False`, indicates that the child widget left and top sides default to being attached to the left and top of the attached dialog box. If `True`, the child widget sides default to being attached to the left and top of the attached dialog box.

DwtNfractionBase
: Specifies the denominator used in specifying fractional positioning.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                            int reason;
                            XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRMap`	The attached dialog box is about to be mapped.
`DwtCRHelpRequested`	The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtBeginCopyToClipboard – Sets up storage and data structures to receive clipboard data.

Syntax

int DwtBeginCopyToClipboard(*display*, *window*, *clip_label*,
widget, *callback*, *item_id*)
Display **display*;
Window *window*;
DwtCompString *clip_label*;
Widget *widget*;
VoidProc *callback*;
long **item_id*;

Arguments

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

clip_label Specifies the label to be associated with the data item. This argument is used to identify the data item, for example, in a clipboard viewer. An example of a label is the name of the application that places the data in the clipboard.

widget Specifies the ID of the widget that will receive messages requesting data previously passed by name. This argument must be present in order to pass data by name. Any valid widget ID in your application can be used. All message handling is done by the cut and paste functions.

callback Specifies the address of the callback function that is called when the clipboard needs data that was originally passed by name. This is also the callback to receive the DELETE message for items that were originally passed by name. This argument must be present in order to pass data by name.

item_id Specifies the number assigned to this data item. The application uses this number in calls to `DwtCopyToClipboard`, `DwtEndCopyToClipboard`, and `DwtCancelCopyToClipboard`.

Description

The `DwtBeginCopyToClipboard` function sets up storage and data structures to receive clipboard data. An application calls `DwtBeginCopyToClipboard` during a cut or copy operation. The data item that these structures receive then becomes the next-paste item in the clipboard.

The *widget* and *callback* arguments must be present in order to pass data by name. The callback format is as follows:

function name(*widget*, *data_id*, *private_id*, *reason*)
Widget **widget*;
int **data_id*;
int **private_id*;
int **reason*;

widget Specifies the ID of the widget passed to `DwtBeginCopyToClipboard`.

data_id Specifies the identifying number returned by `DwtCopyToClipboard`, which identifes the pass-by-name data.

private_id Specifies the private information passed to `DwtCopyToClipboard`.

reason Specifies the reason, which is either `DwtCRClipboardDataDelete` or `DwtCRClipboardDataRequest`.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtCopyToClipboard (3Dwt), DwtEndCopyToClipboard (3Dwt), DwtCancelCopyToClipboard (3Dwt), DwtStartCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCancelCopyFormat (3Dwt)

Name

DwtCancelCopyFormat – Indicates that the application will no longer supply a data item to the clipboard that the application had previously passed by name.

Syntax

int DwtCancelCopyFormat(*display*, *window*, *data_id*)
Display **display*;
Window *window*;
int *data_id*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
data_id	Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This was assigned to the item when it was originally passed by `DwtCopyToClipboard`.

Description

The `DwtCancelCopyFormat` function indicates that the application will no longer supply a data item to the clipboard that the application had previously passed by name.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCancelCopyToClipboard (3Dwt)

Name

DwtCancelCopyToClipboard – Cancels the copy to clipboard that is in progress.

Syntax

void DwtCancelCopyToClipboard(*display*, *window*, *item_id*)
Display **display*;
Window *window*;
long *item_id*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
item_id	Specifies the number assigned to this data item. This number was returned by a previous call to `DwtBeginCopyToClipboard`.

Description

The `DwtCancelCopyToClipboard` function cancels the copy to clipboard that is in progress. `DwtCancelCopyToClipboard` also frees up temporary storage. If `DwtCancelCopyToClipboard` is called, then `DwtEndCopyToClipboard` does not have to be called. A call to `DwtCancelCopyToClipboard` is valid only after a call to `DwtBeginCopyToClipboard` and before a call to `DwtEndCopyToClipboard`.

See Also

DwtBeginCopyToClipboard (3Dwt), DwtEndCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCautionBox, DwtCautionBoxCreate – Creates a caution box widget for the application to display caution messages.

Syntax

Widget DwtCautionBox(*parent_widget*, *name*, *default_position*,
x, *y*, *style*, *label*,
yeslabel, *nolabel*, *cancel_label*,
default_push_button, *callback*,
help_callback)
Widget *parent_widget*;
char **name*;
Boolean *default_position*;
Position *x*, *y*;
int *style*;
DwtCompString *label*;
DwtCompString *yeslabel*;
DwtCompString *nolabel*;
DwtCompString *cancel_label*;
int *default_push_button*;
DwtCallbackPtr *callback*, *help_callback*;

Widget DwtCautionBoxCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, causes `DwtNx` and `DwtNy` to be ignored and forces the default widget position. The default widget position is centered in the parent window. If `False`, the specified `DwtNx` and `DwtNy` attributes are used to position the widget. This

argument sets the DwtNdefaultPosition attribute associated with DwtDialogBoxCreate.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNx core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNy core widget attribute.

style Specifies the style of the caution box widget. You can pass DwtModal (modal) or DwtModeless (modeless). This argument sets the DwtNstyle attribute associated with DwtDialogBoxPopupCreate.

label Specifies the text in the message line or lines. This argument sets the DwtNlabel attribute associated with DwtCautionBoxCreate.

yeslabel Specifies the label for the Yes push button. If the label is a zero length string, the button is not displayed. This argument sets the DwtNyesLabel attribute associated with DwtCautionBoxCreate.

nolabel Specifies the label for the No push button. If the label is a zero length string, the button is not displayed. This argument sets the DwtNnoLabel attribute associated with DwtCautionBoxCreate.

cancel_label Specifies the label for the Cancel push button. If the label is a NULL string, the button is not displayed. This argument sets the DwtNcancelLabel attribute associated with DwtCautionBoxCreate.

default_push_button
Specifies the push button that represents the default user action. You can pass DwtYesButton, DwtNoButton, or DwtCancelButton. This argument sets the DwtNdefaultPushbutton attribute associated with DwtCautionBoxCreate.

callback Specifies the callback function or functions called when the user activates the Yes, No, or Cancel buttons. This argument sets the DwtNyesCallback, DwtNnoCallback, and

DwtNcancelCallback attributes associated with DwtCautionBoxCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtCautionBox and DwtCautionBoxCreate functions create a caution box widget and return its associated widget ID. When calling DwtCautionBox, you set the caution box widget attributes presented in the formal parameter list. For DwtCautionBoxCreate, however, you specify a list of attribute name/value pairs that represent all the possible caution box widget attributes.

A caution box warns the user of the application of the consequences of carrying out an action. It stops application activity and requires the user to provide instructions on how to proceed. Your application should generate a caution box when an action by the user could destroy data or cause a simialr irrevocable event. The caution box usually contains Yes, No, and Cancel push buttons. When possible, caution messages should be written as inquiries. In all cases, the default push button should be the least destructive operation. If DwtNstyle is DwtModal when the user activates any push button, the widget is cleared from the screen, but not destroyed. You can redisplay the widget by calling XtManageChild.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager

DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	5 pixels
DwtNheight	Dimension	5 pixels
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Dialog Pop-Up Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	NOT SUPPORTED	
DwtNunits	NOT SUPPORTED	
DwtNtitle	DwtCompString	Widget name
DwtNstyle	unsigned char	DwtModal
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	NOT SUPPORTED	
DwtNmarginWidth	Dimension	12 pixels
DwtNmarginHeight	Dimension	10 pixels
DwtNdefaultPosition	Boolean	False

DwtNchildOverlap	NOT SUPPORTED	
DwtNresize	unsigned char	DwtResizeShrinkWrap
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	NOT SUPPORTED	
DwtNcancelButton	NOT SUPPORTED	

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNyesLabel	DwtCompString	"Yes"
DwtNnoLabel	DwtCompString	"No"
DwtNcancelLabel	DwtCompString	"Cancel"
DwtNdefaultPushbutton	unsigned char	DwtYesButton
DwtNyesCallback	DwtCallbackPtr	NULL
DwtNnoCallback	DwtCallbackPtr	NULL
DwtNcancelCallback	DwtCallbackPtr	NULL

DwtNlabel — Specifies the text in the message line or lines.

DwtNyesLabel — Specifies the label for the Yes push button. If the label is a zero length string, the button is not displayed.

DwtNnoLabel — Specifies the label for the No push button. If the label is a zero length string, the button is not displayed.

DwtNcancelLabel — Specifies the label for the Cancel push button. If the label is a NULL string, the button is not displayed.

DwtNdefaultPushbutton — Specifies the push button that represents the default user action. You can pass DwtYesButton, DwtNoButton, or DwtCancelButton.

DwtNyesCallback — Specifies the callback function or functions called when the user clicks on the Yes button. For this callback, the reason is DwtCRYes.

DwtNnoCallback — Specifies the callback function or functions called when the user clicks on the No button. For this callback, the reason is DwtCRNo.

DwtNcancelCallback — Specifies the callback function or functions called when the user clicks on the Cancel button. For this callback, the reason is DwtCRCancel.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRYes	The user activated the Yes button.
DwtCRNo	The user activated the No button.
DwtCRCancel	The user activated the Cancel button.
DwtCRFocus	The caution box has received the input focus.
DwtCRHelpRequested	The user selected Help somewhere in the caution box.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtClipboardLock (3Dwt)

Name

DwtClipboardLock – Locks the clipboard from access by other applications.

Syntax

int DwtClipboardLock(*display*, *window*)
Display **display*;
Window *window*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

Description

The `DwtClipboardLock` function locks the clipboard from access by another application until you call `DwtClipboardUnlock`. All clipboard functions lock and unlock the clipboard to prevent simultaneous access. The `DwtClipboardLock` and `DwtClipboardUnlock` functions allow the application to keep the clipboard data from changing between calls to the inquire functions and other clipboard functions. The application does not need to lock the clipboard between calls to `DwtBeginCopyToClipboard` and `DwtEndCopyToClipboard`.

If the clipboard is already locked by another application, `DwtClipboardLock` returns an error status.

Multiple calls to `DwtClipboardLock` by the same application increase the lock level.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtClipboardUnlock (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtClipboardRegisterFormat (3Dwt)

Name

DwtClipboardRegisterFormat – Registers the length of the data for formats not specified by ICCCM conventions.

Syntax

int DwtClipboardRegisterFormat(*display*, *format_name*,
format_length)
Display **display*;
char * *format_name*;
unsigned long *format_length*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
format_name	Specifies a name string for the format. See the table of Data Format Names for the formats defined by ICCCM conventions.
format_length	Specifies the format length in bits: 8, 16, or 32.

Description

The `DwtClipboardRegisterFormat` function allows an application to register the data length for formats not specified by the ICCCM conventions. Failure to register the length of the data results in applications being incompatible across platforms that have different byte-swapping orders.

The following table lists the formats defined by the conventions:

Format Name	Format Length	Description
TARGETS	32	List of valid target atoms
MULTIPLE	32	Look in the ConvertSelection property
TIMESTAMP	32	Timestamping used to acquire selection
STRING	8	ISO Latin 1 (+TAB+NEWLINE) text
TEXT	8	Text in owner's encoding

LIST_LENGTH	32	Number of disjoint parts of selection
PIXMAP	32	Pixmap ID
DRAWABLE	32	Drawable ID
BITMAP	32	Bitmap ID
FOREGROUND	32	Pixel value
BACKGROUND	32	Pixel value
COLORMAP	32	Colormap ID
ODIF	8	ISO Office Document Interchange Format
OWNER_OS	8	Operating system of owner
FILE_NAME	8	Full path name of a file
HOST_NAME	8	See WM_CLIENT_MACHINE
CHARACTER_POSITION	32	Start and end of selection in bytes
LINE_NUMBER	32	Start and end line numbers
COLUMN_NUMBER	32	
LENGTH	32	Number of bytes in selection
USER	8	Name of user running owner
PROCEDURE	8	Name of selected procedure
MODULE	8	Name of selected module
PROCESS	32 or 8	Process ID of owner
TASK	32 or 8	Task ID of owner
CLASS	8	Class of owner–See WM_CLASS
NAME	8	Name of owner–See WM_NAME
CLIENT_WINDOW	32	Top-level window of owner

For information on the built-in selection property names WM_CLIENT_MACHINE, WM_CLASS, and WM_NAME, see the *Guide to the Xlib Library: C Language Binding*.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

`ClipboardBadFormat`	The function failed because the *format_name* or *format_length* was inappropriate. A NULL *format_name* or a *format_length* other than 8, 16, or 32, for example, would be inappropriate.
`ClipboardFail`	The function failed because the application tried to redefine a predefined format. See the table of Data Format Names for the predefined formats.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtClipboardUnlock – Unlocks the clipboard, enabling other applications to access it.

Syntax

int DwtClipboardUnlock(*display*, *window*, *remove_all_locks*)
Display **display*;
Window *window*;
Boolean *remove_all_locks*;

Arguments

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

remove_all_locks
Specifies a boolean value that, when `True`, indicates that all nested locks should be removed. If `False`, indicates that only one level of lock should be removed.

Description

The `DwtClipboardUnlock` function unlocks the clipboard, enabling it to be accessed by other applications.

If multiple calls to `DwtClipboardLock` have occurred, then the same number of calls to `DwtClipboardUnlock` is necessary to unlock the clipboard, unless the *remove_all_locks* argument is `True`.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess` The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtClipboardLock (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCloseHierarchy – Closes a UID hierarchy.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtCloseHierarchy(hierarchy_id)
    DRMHierarchy hierarchy_id;
```

Arguments

hierarchy_id	Specifies the ID of a previously opened UID hierarchy. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.

Description

The `DwtCloseHierarchy` function closes a UID hierarchy previously opened by `DwtOpenHierarchy`. All files associated with the hierarchy are closed by DRM and all associated memory is returned.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMFailure`	The function failed.

See Also

DwtOpenHierarchy(3Dwt)

DwtColorMixCreate (3Dwt)

Name

DwtColorMixCreate – Creates a color mixing widget, which is a pop-up dialog box containing a default color display subwidget and a default color mixer subwidget.

Syntax

Widget DwtColorMixCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtColorMixCreate` function creates a color mixing widget and returns its associated widget ID. Note that unlike most of the other widgets in the XUI toolkit, a color mixing widget cannot be created with a high-level function. When calling `DwtColorMixCreate`, you specify a list of attribute name/value pairs that represents all the possible color mixing widget attributes.

The color mixing widget is a composite widget; that is, it is composed of a parent widget and several child widgets at creation time. The parent widget is a pop-up dialog box that has some labels, handles geometry management, calls back to the application and contains the following child widgets by default:

- A color display subwidget that displays the colors being mixed
- A color mixer subwidget that allows the user to specify colors

- An optional work area widget

While the color mixing widget contains these three child widgets by default, the application can replace either or both the color display and color mixer subwidgets. Thus, applications can provide any type of color display or color mixer tool model.

The default color display widget displays both the original color (the color value supplied by the application when the mixing began) and the current new color. Applications can set the following values:

- The original color values for red, green, and blue
- The new color values for red, green, and blue
- The background color of the display widget
- The dimensions of the color display windows and background area

If the display device is a gray scale, pseudo color, or static color device, the color display widget allocates a maximum of three color cells whenever it becomes managed. If fewer than three color cells are available, the order of precedence is as follows:

1 Original color cell

2 New color cell

3 Background color cell

These color cells are deallocated whenever the widget becomes unmanaged.

If an application replaces the default color display subwidget, the application may provide a function to allow the color mixing widget to pass the current new color value from the color mixer subwidget. Otherwise, the color mixing widget cannot inform the color display subwidget of color changes. The application can return to the default color display subwidget at any time by using `XtSetValues` to set `DwtNdisplayWindow` to NULL.

The default RGB color mixer subwidget provides three scales, each of which represents a percentage of red, green, and blue. Users may also type in the actual X color values (0 to 65535) in the entry fields. When color mixing begins, the color mixer subwidget is set to the current new color values.

If an application replaces the default color mixer subwidget, the new color mixer subwidget must inform the color mixing widget of changes to the current color value. The fastest way to do this is to call the convenience function `DwtColorMixSetNewColor`, although you can also use

XtSetValues. The application can return to the default color mixer subwidget at any time by using XtSetValues to set DwtNmixerWindow to NULL.

Note that setting DwtNdisplayWindow and DwtNmixerWindow to NULL when the color mixing widget is created results in no color display subwidget and no color mixer subwidget. Setting these attributes to NULL after the color mixing widget is created results in returning to the default color display and color mixer subwidgets.

The color mixing widget runs on any XUI display device. On gray scale devices, the default color display subwidget shows the RGB values in gray scale. On static gray (monochrome) devices, the default color display subwidget is not visible.

As far as geometry management is concerned, the color mixing widget conforms to the size of its children.

As far as resizing is concerned, the color mixing widget uses the dialog box shrink wrap mode. It expands and shrinks relative to the size of its children.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Zero pixels
DwtNheight	Dimension	Zero pixels
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL

DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Dialog Box Pop-Up Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNunits	unsigned char	DwtFontUnits
DwtNstyle	unsigned char	DwtModeless
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	XtTranslations	NULL
DwtNmarginWidth	Dimension	10 pixels
DwtNmarginHeight	Dimension	10 pixels
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeShrinkWrap
DwtNnoResize	Boolean	True
DwtNtitle	DwtCompString	"Color Mixing"
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNautoUnmanage	Boolean	False
DwtNdefaultButton	Widget	NULL
DwtNcancelButton	Widget	NULL
DwtNgrabKeySyms	KeySym	The default array contains the Tab key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT()\n\ Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()";

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNmainLabel	DwtCompString	NULL
DwtNdisplayLabel	DwtCompString	NULL
DwtNmixerLabel	DwtCompString	NULL
DwtNorigRedValue	unsigned short	Zero
DwtNorigGreenValue	unsigned short	Zero
DwtNorigBlueValue	unsigned short	Zero
DwtNnewRedValue	unsigned short	Zero, unless DwtNmatchColors is True, in which case DwtNnewRedValue is set to match DwtNorigRedValue whenever the widget is created and mapped.
DwtNnewGreenValue	unsigned short	Zero, unless DwtNmatchColors is True, in which case DwtNnewGreenValue is set to match DwtNorigGreenValue whenever the widget is created and mapped.
DwtNnewBlueValue	unsigned short	Zero, unless DwtNmatchColors is True, in which case DwtNnewBlueValue is set to match DwtNorigBlueValue whenever the widget is created and mapped.
DwtNdisplayWindow	Widget	The color mixing widget display subwidget
DwtNsetNewColorProc	char *	The function used by the color mixing widget to update the new color values displayed in the color display subwidget.
DwtNmixerWindow	Widget	The color mixing widget's RGB color mixer subwidget
DwtNworkWindow	Widget	NULL
DwtNokLabel	DwtCompString	"OK"
DwtNapplyLabel	DwtCompString	"Apply"
DwtNresetLabel	DwtCompString	"Reset"

DwtNcancelLabel	DwtCompString	"Cancel"
DwtNokCallback	DwtCallbackPtr	NULL
DwtNapplyCallback	DwtCallbackPtr	NULL
DwtNcancelCallback	DwtCallbackPtr	NULL
DwtNmatchColors	Boolean	True This attribute can be set only if the default color display widget is used.
DwtNresize	unsigned short	Gray (32767) This attribute can be set only if the default color display widget is used.
DwtNbackGreenValue	unsigned short	Gray (32767) This attribute can be set only if the default color display widget is used.
DwtNbackBlueValue	unsigned short	Gray (32767) This attribute can be set only if the default color display widget is used.
DwtNdisplayColWinWidth	Dimension	80 pixels This attribute can be set only if the default color display widget is used.
DwtNdisplayColWinHeight	Dimension	80 pixels This attribute can be set only if the default color display widget is used.
DwtNdispWinMargin	Dimension	20 pixels This attribute can be set only if the default color display widget is used.
DwtNsliderLabel	DwtCompString	"Percentage" This attribute can be set only if the default color mix tool widget is used.
DwtNvalueLabel	DwtCompString	"Value" This attribute can be set only if the default color mix tool widget is used.
DwtNredLabel	DwtCompString	"Red" This attribute can be set only if the default color mix tool widget is used.

DwtNgreenLabel	DwtCompString	"Green" This attribute can be set only if the default color mix tool widget is used.
DwtNblueLabel	DwtCompString	"Blue" This attribute can be set only if the default color mix tool widget is used.

DwtNmainLabel Specifies the text of the main label, which is centered at the top of the color mixing widget.

DwtNdisplayLabel
Specifies the text of the label centered above the color display widget.

DwtNmixerLabel Specifies the text of the label centered color mixing widget.

DwtNorigRedValue
Specifies the original red color value for the color mixing widget. Applications should set the original red value.

DwtNorigGreenValue
Specifies the original green color value for the color mixing widget. Applications should set the original green value.

DwtNorigBlueValue
Specifies the original blue color value for the color mixing widget. Applications should set the original blue value.

DwtNnewRedValue Specifies the new red color value for the color mixing widget.

DwtNnewGreenValue
Specifies the new green color value for the color mixing widget.

DwtNnewBlueValue
Specifies the new blue color value for the color mixing widget.

DwtNdisplayWindow
Specifies the color display widget. Setting this attribute to NULL at widget creation time causes the

color display widget to not be displayed.

If an application substitutes its own color display widget for the default color display widget, the application is responsible for managing the widget, that is, making it visible and controlling its geometry management. An application can return to the default color display widget by using `XtSetValues` to set this attribute to NULL.

`DwtNsetNewColorProc`
Specifies the function used by the color mixing widget to update the new color values displayed in the color display subwidget. If the application replaces the default color display subwidget and wants the color mixing widget to update the new color, the application must set this attribute. Otherwise, replacing the default color display subwidget sets this attribute to NULL.

`DwtNmixerWindow` Specifies the color mixer subwidget. The default color mixer subwidget is based on the red, green, and blue (RGB) color model. Setting this attribute to NULL at widget creation time causes the color mixer subwidget to not be displayed.

If an application substitutes its own color mixer subwidget for the default color mixer subwidget, the application is responsible for managing the widget, that is, making it visible and controlling its geometry management. An application can later return to the default color mixer subwidget by using `XtSetValues` to set this attribute to NULL.

Applications that use the default color mixer subwidget need not worry about updating the new color. However, applications that provide their own color mixer subwidget are responsible for updating the new color. Applications can do this by using either `XtSetValues` or `DwtColorMixSetNewColor`. Using `DwtColorMixSetNewColor` is recommended because it is more efficient.

`DwtNworkWindow` Specifies an optional work area widget. If this attribute is set and the application manages this

widget, the work window is placed below the color display and color mixer subwidgets (if present) and above the color mixing widget push buttons.

`DwtNokLabel` Specifies the label for the OK push button.

`DwtNapplyLabel` Specifies the label for the Apply push button.

`DwtNresetLabel` Specifies the label for the Reset push button.

`DwtNcancelLabel` Specifies the label for the Cancel push button.

`DwtNokCallback` Specifies the callback function or functions called when the user clicks on the OK push button. For this callback, the reason is `DwtCRActivate`.

`DwtNapplyCallback`
Specifies the callback function or functions called when the user clicks on the Apply push button. For this callback, the reason is `DwtCRApply`.

`DwtNcancelCallback`
Specifies the callback function or functions called when the user clicks on the Cancel button. For this callback, the reason is `DwtCRCancel`.

`DwtNmatchColors` Specifies a boolean value that, when `True`, indicates that the new color values are matched to original color values. If `False`, new color values are not matched to original color values.

This attribute can be set only if the default color display widget is used.

`DwtNbackRedValue`
Specifies the default color display widget's red background color. This attribute can be set only if the default color display widget is used.

`DwtNbackGreenValue`
Specifies the default color display widget's green background color. This attribute can be set only if the default color display widget is used.

`DwtNbackBlueValue`
Specifies the default color display widget's blue background color. This attribute can be set only if the default color display widget is used.

`DwtNdisplayColWinWidth`
: Specifies the width of the original and new color display windows. This attribute can be set only if the default color display widget is used.

`DwtNdisplayColWinHeight`
: Specifies the height of the original and new color display windows. This attribute can be set only if the default color display widget is used.

`DwtNdispWinMargin`
: Specifies the margin between the original and the new color display windows and the edge of the color display widget. The margin is the area affected by the background attributes (set gray by default).

  This attribute can be set only if the default color display widget is used.

`DwtNsliderLabel`
: Specifies the text of the label above the slider representing the RGB scales. This attribute can be set only if the default color mix tool widget is used.

`DwtNvalueLabel`
: Specifies the text of the label above the RGB text entry fields. This attribute can be set only if the default color mix tool widget is used.

`DwtNredLabel`
: Specifies the label for the RGB red scale widget. This attribute can be set only if the default color mix tool widget is used.

`DwtNgreenLabel`
: Specifies the label for the RGB green scale widget. This attribute can be set only if the default color mix tool widget is used.

`DwtNblueLabel`
: Specifies the label for the RGB blue scale widget. This attribute can be set only if the default color mix tool widget is used.

Return Value

This function returns the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        unsigned short newred;
                        unsigned short newgrn;
                        unsigned short newblu;
                        unsigned short origred;
                        unsigned short origgrn;
                        unsigned short origblu;
} DwtColorMixCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRActivate`	The user has activated the OK push button.
`DwtCRApply`	The user has selected the Apply push button.
`DwtCRCancel`	The user has activated the Cancel push button.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

The newred member is set to the new red color value for the color mix widget. The newgrn member is set to the new green color value for the color mix widget. The newblu member is set to the new blue color value for the color mix widget.

The origred member is set to the original red color value for the color mix widget. The origgrn member is set to the original green color value for the color mix widget. The origblu member is set to the original blue color value for the color mix widget.

See Also

DwtColorMixSetNewColor (3Dwt), DwtColorMixGetNewColor (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtColorMixGetNewColor – Returns the red, green, and blue color values to the color mixing widget.

Syntax

void DwtColorMixGetNewColor(*cmw*, *red*, *green*, *blue*)
Widget *cmw*;
unsigned short **red*;
unsigned short **green*;
unsigned short **blue*;

Arguments

cmw	Specifies the widget ID of the color mixing widget.
red	Specifies the current new color red value.
green	Specifies the current new color green value.
blue	Specifies the current new color blue value.
	See the section on colormap functions in the *Guide to the Xlib Library: C Language Binding* for more information on X color values.

Description

The `DwtColorMixGetNewColor` function allows the color mixing widget to pass the current color value created by the color mixer subwidget to the color display subwidget. If the application uses the default color mixer subwidget, using `DwtColorMixGetNewColor` is faster than using `XtGetValues`.

See Also

DwtColorMixSetNewColor (3Dwt), DwtColorMixCreate (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtColorMixSetNewColor (3Dwt)

Name

DwtColorMixSetNewColor – Sets the new red, green, and blue color values in the color mixing widget.

Syntax

void DwtColorMixSetNewColor(*cmw*, *red*, *green*, *blue*)
Widget *cmw*;
unsigned short *red*;
unsigned short *green*;
unsigned short *blue*;

Arguments

cmw	Specifies the widget ID of the color mixing widget.
red	Specifies the new color red value. You can express the value in percentages or by the X color values (0 to 65535).
green	Specifies the new color green value. You can express the value in percentages or by the X color values (0 to 65535).
blue	Specifies the new color blue value. You can express the value in percentages or by the X color values (0 to 65535).
	See the section on colormap functions in the *Guide to the Xlib Library: C Language Binding* for more information on X color values.

Description

The `DwtColorMixSetNewColor` function allows the user-supplied color mixer subwidget to pass the current color values to the color mixing widget. Using `DwtColorMixSetNewColor` is more efficient than using `XtSetValues`.

See Also

DwtColorMixGetNewColor (3Dwt), DwtColorMixCreate (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCommandAppend – Appends the passed string to the current command line and executes it, if required.

Syntax

void DwtCommandAppend(*widget*, *command*)
Widget *widget*;
char **command*;

Arguments

widget	Specifies the ID of the command window widget to whose command line you want to append the passed string.
command	Specifies the text to be appended to the command line. This argument is a NULL-terminated string.

Description

The `DwtCommandAppend` function appends the passed string to the current command line, within the command window widget. If the string sent is terminated with a carriage return (<CR>) or carriage return and/or linefeed (<CR><LF>) character, then the command is executed, the application is informed, the command is moved to the command history, and a new prompt is issued.

See Also

DwtCommandWindow (3Dwt), DwtCommandErrorMessage (3Dwt), DwtCommandSet (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCommandErrorMessage (3Dwt)

Name

DwtCommandErrorMessage – Writes an error message in the command window and refreshes the command line.

Syntax

void DwtCommandErrorMessage(*widget*, *error*)
Widget *widget*;
char **error*;

Arguments

widget	Specifies the ID of the command window widget in whose command window you want to write an error message.
error	Specifies the error message to be placed in the bottom-most history line in the command window widget. This argument is a NULL-terminated string.

Description

The `DwtCommandErrorMessage` function writes an error message in the history area within the command window widget. The history is first scrolled up.

See Also

DwtCommandWindow (3Dwt), DwtCommandAppend (3Dwt), DwtCommandSet (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCommandSet – Replaces the current command string with the one passed and executes it, if required.

Syntax

void DwtCommandSet(*widget*, *command*)
Widget *widget*;
char **command*;

Arguments

widget	Specifies the ID of the command window widget whose current command string you want to replace.
command	Specifies the text to replace the text currently on the command line. This argument is a NULL-terminated string.

Description

The `DwtCommandSet` function replaces the current command string with the passed string within the command window widget. A zero length string is used to clear the current command line. If the string is terminated by a carriage return (<CR>), linefeed (<LF>), or carriage return and/or linefeed (<CR><LF>), then the command is executed, the application is informed, the command is moved to the command history, and a new prompt is issued.

See Also

DwtCommandWindow (3Dwt), DwtCommandAppend (3Dwt), DwtCommandErrorMessage (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCommandWindow(3Dwt)

Name

DwtCommandWindow, DwtCommandWindowCreate – Creates a command window widget.

Syntax

Widget DwtCommandWindow(*parent_widget*, *name*, *prompt*,
lines, *callback*, *help_callback*)
Widget *parent_widget*;
char **name*;
DwtCompString *prompt*;
int *lines*;
DwtCallbackPtr *callback*, *help_callback*;

Widget DwtCommandWindowCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
prompt	Specifies the command line prompt. This argument sets the `DwtNprompt` attribute associated with `DwtCommandWindowCreate`.
lines	Specifies the number of command history lines visible in the command window widget. This argument sets the `DwtNlines` attribute associated with `DwtCommandWindowCreate`.
callback	Specifies the callback function or functions called when the user enters a command or changes the contents of a command line. This argument sets the `DwtNcommandEnteredCallback` and `DwtNvalueChangedCallback` attributes associated with `DwtCommandWindowCreate`.
help_callback	Specifies the callback function or functions called when a

help request is made. This argument sets the DwtNhelpCallback common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtCommandWindow and DwtCommandWindowCreate functions create an instance of a command window widget and return its associated widget ID. The command window widget handles command line entry, command line history, and command line recall. When calling DwtCommandWindow, you set the command window widget attributes presented in the formal parameter list. For DwtCommandWindowCreate, however, you specify a list of attribute name/value pairs that represent all the possible command window widget attributes.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	zero
DwtNheight	Dimension	zero
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True

DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Dialog Pop-Up Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	NOT SUPPORTED	
DwtNunits	NOT SUPPORTED	
DwtNtitle	DwtCompString	Widget name
DwtNstyle	unsigned char	DwtModal
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	NOT SUPPORTED	
DwtNmarginWidth	Dimension	12 pixels
DwtNmarginHeight	Dimension	10 pixels
DwtNdefaultPosition	Boolean	True This causes the command window to be positioned in the bottom left-hand corner of the parent widget.
DwtNchildOverlap	NOT SUPPORTED	
DwtNresize	NOT SUPPORTED	
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNautoUnmanage	Boolean	True

DwtNdefaultButton	NOT SUPPORTED	
DwtNcancelButton	Widget	NULL
DwtNcancelButton	NOT SUPPORTED	

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNvalue	char *	NULL
DwtNprompt	DwtCompString	">"
DwtNlines	short	Two lines
DwtNhistory	char *	""
DwtNcommandEnteredCallback	DwtCallbackPtr	NULL
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNtTranslation	XtTranslations	NULL

DwtNvalue
: Specifies the current contents of the command line string. When a command-entered callback is made, this attribute will be the command line that just executed.

DwtNprompt
: Specifies the command line prompt.

DwtNlines
: Specifies the number of command history lines visible in the command window widget.

DwtNhistory
: Specifies the contents of the command line history. Multiple lines should be separated by a linefeed character (<LF>).

DwtNcommandEnteredCallback
: Specifies the callback function or functions called when the user terminated the command line with a carriage return/line feed. For this callback, the reason is DwtCRCommandEntered.

DwtNvalueChangedCallback
: Specifies the callback function or functions called when the contents of the command line have changed. For this callback, the reason is DwtCRValueChanged.

DwtNtTranslation
: Specifies the translations used for the command line text field.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                            int reason;
                            XEvent *event;
                            int length;
                            char *value;
} DwtCommandWindowCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRCommandEntered`	The user terminated the command line with a carriage return/line feed.
`DwtCRValueChanged`	The contents of the command line have changed.
`DwtCRFocus`	The command window widget has received the input focus.
`DwtCRHelpRequested`	The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*. The length member is set to the length of the current command line contents. The value member is set to the current command line contents.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCopyFromClipboard – Retrieves a data item from the clipboard.

Syntax

int DwtCopyFromClipboard(*display*, *window*, *format_name*,
buffer, *length*,
num_bytes, *private_id*)
Display **display*;
Window *window*;
char **format_name*;
char **buffer*;
unsigned long *length*;
unsigned long **num_bytes*;
int **private_id*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
format_name	Specifies the name of a format in which the data is stored on the clipboard.
buffer	Specifies the buffer to which the application wants the clipboard to copy the data.
length	Specifies the length of the application buffer.
num_bytes	Specifies the number of bytes of data copied into the application buffer.
private_id	Specifies the private data stored with the data item by the application that placed the data item on the clipboard. If the application did not store private data with the data item, this argument returns zero.

DwtCopyFromClipboard (3Dwt)

Description

The `DwtCopyFromClipboard` function retrieves the current next-paste item from clipboard storage.

`DwtCopyFromClipboard` returns a warning under the following circumstances:

- The data needs to be truncated because the buffer length is too short
- The clipboard is locked
- There is no data on the clipboard

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	All data on the clipboard has been copied successfully. A successful copy can be a one-time operation using `DwtCopyFromClipboard` alone, or an incremental operation using multiple calls to `DwtCopyFromClipboard` between calls to `DwtStartCopyFromClipboard` and `DwtEndCopyFromClipboard`.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.
`ClipboardTruncate`	If using `DwtCopyFromClipboard` alone, the data returned is truncated because the user did not provide a buffer that was large enough to hold the data. If using multiple calls to `DwtCopyFromClipboard` in between calls to `DwtStartCopyFromClipboard` and `DwtEndCopyFromClipboard`, more data in the requested format remains to be copied from the clipboard.

`ClipboardNoData`	The function could not find data on the clipboard corresponding to the format requested. This could occur because: (1) the clipboard is empty; (2) there is data on the clipboard but not in the requested format; and (3) the data in the requested format was passed by name and is no longer available.

See Also

DwtStartCopyFromClipboard (3Dwt), DwtEndCopyFromClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCopyToClipboard (3Dwt)

Name

DwtCopyToClipboard – Copies a data item to the clipboard.

Syntax

int DwtCopyToClipboard(*display*, *window*, *item_id*,
format_name, *buffer*, *length*,
private_id, *data_id*)
Display **display*;
Window *window*;
long *item_id*;
char **format_name*;
char **buffer*;
unsigned long *length*;
int *private_id*;
int **data_id*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
item_id	Specifies the number assigned to this data item. This number was returned by a previous call to `DwtBeginCopyToClipboard`.
format_name	Specifies the name of the format in which the data item is stored.
buffer	Specifies the buffer from which the clipboard copies the data.
length	Specifies the length of the data being copied to the clipboard.
private_id	Specifies the private data that the application wants to store with the data item.
data_id	Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This argument is required only for data that is passed by name.

Description

The `DwtCopyToClipboard` function copies a data item to clipboard storage. The data item is not actually entered in the clipboard data structure until the call to `DwtEndCopyToClipboard`. Additional calls to `DwtCopyToClipboard` before a call to `DwtEndCopyToClipboard` add data item formats to the same data item or append data to an existing format.

If the *buffer* argument is NULL, the data is considered passed by name. If data passed by name is later needed by another application, the application that owns the data receives a callback with a request for the data. The application that owns the data must then transfer the data to the clipboard with the `DwtReCopyToClipboard` function. When a data item that was passed by name is deleted from the clipboard, the application that owns the data receives a callback that states that the data is no longer needed.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtEndCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCreateFontList (3Dwt)

Name

DwtCreateFontList – Creates a new font list.

Syntax

DwtFontList DwtCreateFontList(*font*, *charset*)
XFontStruct **font*;
long *charset*;

Arguments

font	Specifies a pointer to a font structure for which the new font list is generated.
charset	Specifies the character set identifier for the font. Values for this argument can be found in the required file `/usr/include/cda_def.h`.

Description

The `DwtCreateFontList` function creates a new font list for the font and character set. It also allocates the space for the font list. The end of the font list is marked by an element whose character set value is -1.

Return Value

This function returns a new font list. However, it returns NULL if the font specified in *font* is NULL.

See Also

DwtAddFontList (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSbytecmp – Determines if two compound-strings are identical.

Syntax

int DwtCSbytecmp(*compound_string1*, *compound_string2*)
DwtCompString *compound_string1*, *compound_string2*;

Arguments

compound_string1
Specifies a compound-string to be compared with *compound_string2*.

compound_string2
Specifies a compound-string to be compared with *compound_string1*.

Description

The `DwtCSbytecmp` function returns zero if *compound_string1* and *compound_string2* are exactly the same (byte to byte). It returns one if they are not the same.

Return Value

Zero if *compound_string1* and *compound_string2* are exactly the same (byte to byte). It returns one if they are not the same.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCSempty (3Dwt)

Name

DwtCSempty – Determines if the compound-string contains any text segments.

Syntax

int DwtCSempty(*compound_string*)
DwtCompString *compound_string*;

Arguments

compound_string
Specifies the compound-string.

Description

The `DwtCSempty` function determines if the compound-string contains any text segments. `DwtCSempty` returns `True` if all text segments in the compound-string are empty. Otherwise, it returns `False`.

Return Value

`DwtCSempty` returns `True` if all text segments in the compound-string are empty. Otherwise, it returns `False`.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSString – Creates a compound-string.

Syntax

DwtCompString DwtCSString(*text*, *charset*, *direction_r_to_l*,
language, *rend*)
char **text*;
unsigned long *charset*;
char *direction_r_to_l*;
unsigned long *language*;
DwtRendMask *rend*;

Arguments

text	Specifies the text string to be converted to a compound-string.
charset	Specifies the character set for the compound-string. Values for this argument can be found in the required file `/usr/include/cda_def.h`.
direction_r_to_l	Specifies the direction in which the text is drawn and wraps. You can pass `DwtDirectionLeftDown` (text is drawn from left to right and wraps down); `DwtDirectionRightUp` (text is drawn from left to right and wraps up); `DwtDirectionLeftDown` (text is drawn from right to left and wraps down); or `DwtDirectionLeftUp` (text is drawn from right to left and wraps up).
language	Included for future use.
rend	Included for future use.

Description

The `DwtCSString` function creates a compound-string from information in the argument list. Space for the resulting string is allocated within the function. After using this function, you should free the space by calling `XtFree`.

Return Value

This function returns the resulting compound-string. However, it returns a NULL pointer if the input string is NULL.

See Also

DwtLatin1String (3Dwt), DwtString (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSText, DwtCSTextCreate – Creates a compound-string text widget.

Syntax

Widget DwtCSText(*parent_widget*, *name*, *x*, *y*, *cols*, *rows*, *value*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
Dimension *cols*, *rows*;
DwtCompString *value*;

Widget DwtCSTextCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the top of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
cols	Specifies the width of the text window measured in character cells. This argument sets the `DwtNcols` attribute associated with `DwtCSTextCreate`.
rows	Specifies the height of the text window measured in character cells or number of lines. This argument sets the `DwtNrows` attribute associated with `DwtCSTextCreate`.
value	Specifies the text contents of the compound-string text widget. This argument sets the `DwtNvalue` attribute associated with `DwtCSTextCreate`.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtCSText` and `DwtCSTextCreate` functions create an instance of a compound-string text widget and return its associated widget ID. When calling `DwtCSText`, you set the compound-string text widget attributes presented in the formal parameter list. For `DwtCSTextCreate`, however, you specify a list of attribute name/value pairs that represent all the possible compound-string text widget attributes. The compound-string text widget enables the application to display a single or multiline field of text for input and editing by the user. By default the text window expands or shrinks as the user enters or deletes text characters. Note that the text window does not shrink below the initial size set at creation time.

The compound-string text widget does not support children; therefore, there is no geometry or resize semantics.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	Set as large as necessary to display the `DwtNrows` and `DwtNcols` with the specified `DwtNmarginWidth` and `DwtNmarginHeight`
`DwtNheight`	`Dimension`	Set as large as necessary to display the `DwtNcols` and `DwtNrows` with the specified `DwtNmarginHeight` and `DwtNmarginWidth`
`DwtNborderWidth`	`Dimension`	One pixel

DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

You can set the following widget-specifc attributes in the *override_arglist*:

Attribute Name	Data Type	Default
DwtNmarginWidth	Dimension	2 pixels
DwtNmarginHeight	Dimension	Two pixels
DwtNcols	Dimension	20 characters
DwtNrows	Dimension	1 character
DwtNtopPosition	DwtTextPosition	Zero
DwtNwordWrap	Boolean	False
DwtNscrollVertical	Boolean	False
DwtNresizeHeight	Boolean	True
DwtNresizeWidth	Boolean	True
DwtNvalue	char *	""
DwtNeditable	Boolean	True
DwtNmaxLength	int	2**31-1
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNlostFocusCallback	DwtCallbackPtr	NULL
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNinsertionPointVisible	Boolean	True
DwtNautoShowInsertPoint	Boolean	True
DwtNinsertionPosition	int	Zero

DwtNforeground	Pixel	The current server's default foreground
DwtNfont	DwtFontList	The current server's DwtFontList
DwtNblinkRate	int	500 milliseconds
DwtNscrollLeftSide	Boolean	False
DwtNhalfBorder	Boolean	True
DwtNpendingDelete	Boolean	True
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNtextPath	int	Left to right
DwtNeditingPath	int	Left to right
DwtNbidirectionalCursor	Boolean	False
DwtNnofontCallback	DwtCallbackPtr	NULL

DwtNmarginWidth Specifies the number of pixels between the left or right edge of the window and the text.

DwtNmarginHeight
Specifies the number of pixels between the top or bottom edge of the window and the text.

DwtNcols Specifies the width of the text window measured in character spaces.

DwtNrows Specifies the height of the text window measured in character heights or number of line spaces.

DwtNtopPosition Specifies the position to display at the top of the window.

DwtNwordWrap Specifies a boolean value that, when True, indicates that lines are broken at word breaks and text does not run off the right edge of the window.

DwtNscrollVertical
Specifies a boolean value that, when True, adds a scroll bar that allows the user to scroll vertically through the text.

DwtNresizeHeight
Specifies a boolean value that, when True, indicates that the compound-string text widget resizes its height to accommodate all the text contained in the widget. If this is set to True, the text will always be displayed starting from the first position in the source, even if instructed otherwise. This attribute is ignored if DwtNscrollVertical is

	True.
DwtNresizeWidth	Specifies a boolean value that, when True, indicates that the compound-string text widget resizes its width to accommodate all the text contained in the widget. This argument is ignored if DwtNwordWrap is True.
DwtNvalue	Specifies the text contents of the compound-string text widget. If you accept the default of NULL, the text path and editing path are set to DwtDirectionRightDown. Otherwise, the text path and editing path are set from the direction of the first segment of the value.
DwtNeditable	Specifies a boolean value that, when True, indicates that the user can edit the text in the compound-string text widget. If False, prohibits the user from editing the text.
DwtNmaxLength	Specifies the maximum length of the text string, in characters, in the compound-string text widget.
DwtNfocusCallback	Specifies the callback function or functions called when the compound-string text widget accepted the input focus. For this callback, the reason is DwtCRFocus.
DwtNhelpCallback	Specifies the callback function or functions called when a help request is made. For this callback, the reason is DwtCRHelpRequested.
DwtNlostFocusCallback	Specifies the callback function or functions called when the compound-string text widget loses input focus. For this callback, the reason is DwtCRLostFocus.
DwtNvalueChangedCallback	Specifies the callback function or functions called when the value of the compound-string text widget changes. For this callback, the reason is DwtCRValueChanged.
DwtNinsertionPointVisible	

Specifies a boolean value that, when `True`, indicates that the insertion point is marked by a blinking text cursor.

`DwtNautoShowInsertPoint`
Specifies a boolean value that, when `True`, ensures that the text visible in the compound-string text widget window contains the insertion point. This means that if the insertion point changes, the contents of the compound-string text widget window might scroll in order to bring the insertion point into the window.

`DwtNinsertionPosition`
Specifies the current location of the insertion point.

`DwtNforeground` Specifies the pixel for the foreground of the compound-string text widget.

`DwtNfont` Specifies the font list to be used for the compound-string text widget.

`DwtNblinkRate` Specifies the blink rate of the text cursor in milliseconds.

`DwtNscrollLeftSide`
Specifies a boolean value that, when `True`, indicates that the vertical scroll bar should be placed on the left side of the compound-string text window. This attribute is ignored if `DwtNscrollVertical` is `False`.

`DwtNhalfBorder` Specifies a boolean value that, when `True`, indicates that a border is displayed only on the starting edge and bottom edge of the compound-string text widget.

`DwtNpendingDelete`
Specifies a boolean value that, when `True`, indicates that selected text containing the insertion point is deleted when new text is entered.

`DwtNdirectionRToL`
Specifies the direction in which the text is drawn and wraps. You can pass `DwtDirectionLeftDown` (text is drawn from left to right and wraps down); `DwtDirectionRightUp` (text is drawn from left

to right and wraps up); DwtDirectionLeftDown (text is drawn from right to left and wraps down); or DwtDirectionLeftUp (text is drawn from right to left and wraps up). The DwtNdirectionRToL attribute only affects the direction toward which the window is resized.

DwtNtextPath — Specifies a read-only value that holds the main text path (direction) of the text in the compound-string text widget. It is set from the initial compound-string value of the widget. This attribute is used only if DwtNvalue is NULL.

DwtNeditingPath — Specifies a read-only value that holds the current editing text path (direction) in the compound-string text widget. It is set initially equal to DwtNtextPath. This attribute is used only if DwtNvalue is NULL.

DwtNbidirectionalCursor — Specifies a boolean value that, when True, indicates that the shape of the cursor at the insertion point will be dependent on the current editing direction.

DwtNnofontCallback — Specifies a callback function called when the compound-string text widget has failed to find a font needed for the display of a text tagged by a specific character set. For this callback, the reason is DwtCRNoFont.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        char *charset;
                        unsigned int charset_len;
} DwtCSTextCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRFocus`	The compound-string text widget has received the input focus.
`DwtCRLostFocus`	The compound-string text widget has lost the input focus.
`DwtCRValueChanged`	The user changed the value of the text in the compound-string text widget.
`DwtCRHelpRequested`	The user selected Help.
`DwtCRNoFont`	The widget font list contained no entry for the required character set.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding.*

The charset member is set to the character set ID for which the widget has no matching font in its font list. The callback should modify the widget font list to include an entry for the required character set.

The charset_len member is set to the length of the charset string.

See Also

DwtCSTextReplace (3Dwt), DwtCSTextGetString (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSTextClearSelection – Clears the global selection highlighted in the compound-string text widget.

Syntax

void DwtCSTextClearSelection(*widget*, *time*)
Widget *widget*;
Time *time*;

Arguments

widget	Specifies the ID of the compound-string text widget.
time	Specifies the time of the event that led to the call to `XSetSelectionOwner`. You can pass either a timestamp or `CurrentTime`. Whenever possible, however, use the timestamp of the event leading to the call.

Description

The `DwtCSTextClearSelection` function clears the global selection highlighted in the compound-string text widget.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextGetString (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCSTextGetEditable (3Dwt)

Name

DwtCSTextGetEditable – Obtains the current edit permission state indicating whether the user can edit the text in the compound-string text widget.

Syntax

Boolean DwtCSTextGetEditable(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the compound-string text widget.

Description

The `DwtCSTextGetEditable` function returns the current edit-permission-state, which indicates whether the user can edit the text in the compound-string text widget. If the function returns `True`, the user can edit the string text in the compound-string text widget. If it returns `False`, the user cannot edit the text.

Return Value

Specifies a boolean value that, when `True`, indicates the user can edit the text in the compound string text widget. When `False`, the user cannot edit the text.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSTextGetMaxLength – Obtains the current maximum allowable length of the text in the compound-string text widget.

Syntax

int DwtCSTextGetMaxLength(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the compound-string text widget.

Description

The `DwtCSTextGetMaxLength` function returns the current maximum allowable length of the text in the compound-string text widget.

Return Value

This function returns the maximum length, in characters, of the text in the compound string text widget.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextGetString (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCSTextGetSelection (3Dwt)

Name

DwtCSTextGetSelection – Retrieves the global selection, if any, currently highlighted, in the compound string text widget.

Syntax

DwtCompString DwtCSTextGetSelection(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the compound-string text widget.

Description

The `DwtCSTextGetSelection` function retrieves the text currently highlighted (selected) in the compound string text widget. It returns a NULL pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the text by calling `XtFree`.

Return Value

This function returns a pointer to the selected compound string text.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextGetString (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt),
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSTextGetString – Retrieves all text from the compound-string text widget.

Syntax

DwtCompString DwtCSTextGetString(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the compound-string text widget.

Description

The `DwtCSTextGetString` function retrieves the current compound-string from the compound-string text widget. The application is responsible for freeing the storage associated with the string by calling `XtFree`.

Return Value

This function returns a pointer to a compound string holding all the current text in the compound string text widget.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCSTextReplace (3Dwt)

Name

DwtCSTextReplace – Replaces a portion of the current text in the compound-string text widget or inserts some new text into the current text of the compound-string text widget.

Syntax

void DwtCSTextReplace(*widget*, *from_pos*, *to_pos*, *value*)
Widget *widget*;
int *from_pos*, *to_pos*;
DwtCompString *value*;

Arguments

widget	Specifies the ID of the compound-string text widget.
from_pos	Specifies the first character position of the compound-string text being replaced.
to_pos	Specifies the last character position of the compound-string text being replaced.
value	Specifies the text to replace part of the current text in the compound-string text widget.

Description

The `DwtCSTextReplace` function replaces part of the text in the compound-string text widget. Within the widget, positions are numbered starting at 0 and increasing sequentially. For example, to replace the second and third characters in the text, *from_pos* should be 1 and *to_pos* should be 3. To insert text after the fourth character, *from_pos* and *to_pos* should both be 4.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSTextSetEditable – Sets the permission state that determines whether the user can edit text in the compound-string text widget.

Syntax

void DwtCSTextSetEditable(*widget*, *editable*)
Widget *widget*;
Boolean *editable*;

Arguments

widget	Specifies the ID of the compound-string text widget.
editable	Specifies a boolean value that, when `True`, indicates that the user can edit the text in the compound-string text widget. If `False`, prohibits the user from editing the text.

Description

The `DwtCSTextSetEditable` function sets the edit permission state information concerning whether the user can edit text in the compound-string text widget.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCSTextSetMaxLength (3Dwt)

Name

DwtCSTextSetMaxLength – Sets the maximum allowable length of the text in the compound-string text widget.

Syntax

void DwtCSTextSetMaxLength(*widget*, *max_length*)
Widget *widget*;
int *max_length*;

Arguments

widget	Specifies the ID of the compound-string text widget.
max_length	Specifies the maximum length, in characters, of the text in the compound string text widget. This argument sets the `DwtNmaxLength` attribute associated with `DwtCSTextCreate`.

Description

The `DwtCSTextSetMaxLength` function sets the maximum allowable length of the text in the compound-string text widget and prevents the user from entering text longer than this limit.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCSTextSetSelection – Highlights the specified text in the compound-string text widget and makes it the current global selection.

Syntax

void DwtCSTextSetSelection(*widget*, *first*, *last*, *time*)
Widget *widget*;
int *first*, *last*;
Time *time*;

Arguments

widget	Specifies the ID of the compound-string text widget.
first	Specifies the first character position of the selected compound-string text.
last	Specifies the last character position of the selected compound-string text.
time	Specifies the time of the event that led to the call to `XSetSelectionOwner`. You can pass either a timestamp or `CurrentTime`. Whenever possible, however, use the timestamp of the event leading to the call.

Description

The `DwtCSTextSetSelection` function makes the specified text in the compound-string text widget the current global selection and highlights it in the compound-string text widget. Within the text window, *first* marks the first character position and *last* marks the last position. The field characters start at 0 and increase sequentially.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextGetString (3Dwt), DwtCSTextSetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCSTextSetString (3Dwt)

Name

DwtCSTextSetString – Changes the text in the compound-string text widget.

Syntax

void DwtCSTextSetString(*widget*, *value*)
Widget *widget*;
DwtCompString *value*;

Arguments

widget	Specifies the ID of the compound-string text widget.
value	Specifies the text that replaces all text in the current compound-string text widget.

Description

The `DwtCSTextSetString` function completely changes the text in the compound-string text widget.

See Also

DwtCSText (3Dwt), DwtCSTextCreate (3Dwt), DwtCSTextReplace (3Dwt), DwtCSTextGetString (3Dwt), DwtCSTextGetEditable (3Dwt), DwtCSTextSetEditable (3Dwt), DwtCSTextGetMaxLength (3Dwt), DwtCSTextSetMaxLength (3Dwt), DwtCSTextSetSelection (3Dwt), DwtCSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCStrcat, DwtCStrncat – Appends a copy of a compound-string to the end of another compound-string.

Syntax

DwtCompString DwtCStrcat(*compound_string1*,
compound_string2)
DwtCompString *compound_string1*, *compound_string2*;

DwtCompString DwtCStrncat(*compound_string1*,
compound_string2,
num_chars)
DwtCompString *compound_string1*, *compound_string2*;
int *num_chars*;

Arguments

compound_string1
Specifies a compound-string to which a copy of *compound_string2* is appended.

compound_string2
Specifies a compound-string that is appended to the end of *compound_string1*.

num_chars Specifies the number of characters to be appended to the specified compound-string. If *num_chars* is less than the length of *compound_string2*, the resulting string will not be a valid compound-string.

Description

The `DwtCStrcat` function appends *compound_string2* to the end of *compound_string1* and returns the resulting string. The original strings are preserved. The space for the resulting compound-string is allocated within the function. After using this function, you should free this space by calling `XtFree`.

The `DwtCStrncat` function appends no more than the number of characters specified in *num_chars*, which includes tag and length sections of the compound-string.

Return Value

These functions return a pointer to the resulting compound-string.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtCStrcpy, DwtCStrncpy – Copies a compound-string.

Syntax

DwtCompString DwtCStrcpy(*compound_string1*)
DwtCompString *compound_string1*;

DwtCompString DwtCStrncpy(*compound_string1*, *num_chars*)
DwtCompString *compound_string1*;
int *num_chars*;

Arguments

compound_string1
Specifies a compound-string to be copied to the output string.

num_chars Specifies the number of characters to be copied. If *num_chars* is less than the length of *compound_string1*, the resulting string will not be a valid compound-string.

Description

The `DwtCStrcpy` function copies the string in *compound_string1*.

The `DwtCStrncpy` function copies exactly the number of characters specified in *num_chars*, including the headers and trailers.

The space for the resulting compound-string is allocated with these functions. After using these functions, you should free this space by calling `XtFree`.

Return Value

These functions return a pointer to the resulting compound-string.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtCStrlen (3Dwt)

Name

DwtCStrlen – Returns the number of bytes in a compound-string.

Syntax

int DwtCStrlen(*compound_string1*)
DwtCompString *compound_string1*;

Arguments

compound_string1
Specifies a compound-string whose length is determined.

Description

The `DwtCStrlen` function returns the number of bytes in *compound_string1*, including compound-string terminators for headers and trailers. If the compound-string has an invalid stucture, zero is returned.

Return Value

This function returns the number of bytes in *compound_string1*, including compound-string terminators for headers and trailers. If the compound-string has an invalid stucture, zero is returned.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtDialogBox, DwtDialogBoxCreate, DwtDialogBoxPopupCreate – Creates a dialog box widget to contain other subwidgets.

Syntax

Widget DwtDialogBox(*parent_widget*, *name*, *default_position*,
x, *y*, *title*, *style*,
map_callback, *help_callback*)
Widget *parent_widget*;
char **name*;
Boolean *default_position*;
Position *x*, *y*;
DwtCompString *title*;
unsigned char *style*;
DwtCallbackPtr *map_callback*, *help_callback*;

Widget DwtDialogBoxCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Widget DwtDialogBoxPopupCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, causes `DwtNx` and `DwtNy` to be ignored and forces the default widget position. The default widget position is centered in the parent window. If `False`, the specified `DwtNx` and `DwtNy` attributes are used to position the widget.

If the dialog box is displayed partially off the screen as a result of being centered in the parent window, the centering rule is violated. When this occurs, the parent window is repositioned so that the entire dialog box is displayed on the screen.

The pop-up dialog box is recentered every time it is popped up. Consequently, if the parent moves in between invocations of the dialog box, the box pops up centered in the parent window's new location. However, the dialog box does not dynamically follow its parent while it is displayed. If the parent is moved, the dialog box will not move until the next time it is popped up.

If the user moves the dialog box with the window manager, the toolkit turns off `DwtNdefaultPosition`. This results in the dialog box popping up in the location specified by the user on each subsequent invocation. This argument sets the `DwtNdefaultPosition` attribute associated with `DwtDialogBoxCreate`.

x — Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.

y — Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.

title — Specifies the compound-string label. The label is given to the window manager for the title bar if `DwtNstyle` is `DwtModeless`. This argument sets the `DwtNtitle` attribute associated with `DwtDialogBoxPopupCreate`.

style — Specifies the style of the dialog box widget. You can pass `DwtModal`, `DwtModeless`, or `DwtWorkarea`. You cannot change `DwtNstyle` after the widget is created. This argument sets the `DwtNstyle` attribute associated with `DwtDialogBoxCreate` or `DwtDialogBoxPopupCreate`.

map_callback — Specifies the callback function or functions called when the window is about to be mapped. For this callback, the reason is `DwtCRMap`. Note that *map_callback* is supported only if

style is DwtModal or DwtModeless. If *style* is DwtWorkarea, *map_callback* is ignored.

This argument sets the DwtNmapCallback attribute associated with DwtDialogBoxPopupCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

Depending on the constant you pass to DwtNstyle, the DwtDialogBox function creates a dialog box or a pop-up dialog box widget. The DwtDialogBoxCreate function creates a dialog box widget, and DwtDialogBoxPopupCreate creates a pop-up dialog box widget. Upon completion, these functions return the associated widget ID. When calling DwtDialogBox, you set the dialog box widget attributes presented in the formal parameter list. For DwtDialogBoxCreate and DwtDialogBoxPopupCreate, however, you specify a list of attribute name/value pairs that represent all the possible dialog box widget attributes.

The dialog box widget is a composite widget that contains other subwidgets. Each subwidget displays information or requests and/or handles input from the user.

The dialog box widget functions as a container only, and provides no input semantics over and above the expressions of the widgets it contains.

Subwidgets can be positioned within the dialog box in two ways: by font units and by pixel units. By default, subwidgets are positioned in terms of font units (that is, DwtNunits is DwtFontUnits). The X font units are defined to be one-fourth the width of whatever font is supplied for the common attribute DwtNfont. The Y font units are defined to be one-eighth the width of whatever font is supplied for DwtNfont. (Width is taken from the QUAD_WIDTH property of the font.) Subwidgets can also be positioned in terms of pixel units (that is, DwtNunits is DwtPixelUnits).

Note that when changing DwtNtextMergeTranslations, the existing widgets are not affected. The new value for DwtNtextMergeTranslations acts only on widgets that are added after the pop-up dialog box is created.

Pop-up dialog box widgets create their own shells as parents. Therefore, to set the colormap of a pop-up dialog box, you must set the colormap of its parent shell. (To find the parent shell, use XtParent.) For nonpop-up widgets, the shell widget ID is returned from XtInitialize. You need only set the colormap once on the returned shell widget.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Set as large as necessary to hold all child widgets
DwtNheight	Dimension	Set as large as necessary to hold all child widgets
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNunits	unsigned char	DwtFontUnits
DwtNstyle	unsigned char	For DwtDialogBoxCreate, the default is DwtWorkarea. For DwtDialogBoxPopupCreate, the default is DwtModeless.
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	XtTranslations	NULL
DwtNmarginWidth	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreate, the default is 3 pixels.
DwtNmarginHeight	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreate, the default is 3 pixels.
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeGrowOnly
DwtNgrabKeySyms	KeySym	The default array contains the Tab key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT()\n\ Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()";

The following table lists the widget-specific attributes for the pop-up dialog box widget.

Attribute Name	Data Type	Default
DwtNtitle	DwtCompString	When DwtNstyle is DwtModal, the default is NULL When DwtNstyle is DwtModeless, the default is the widget name
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	Widget	NULL
DwtNcancelButton	Widget	NULL
DwtNautoUnrealize	Boolean	False

DwtNforeground
: Specifies the color of foreground gadget children in the widget window.

DwtNhighlight
: Specifies the color used for highlighting gadget children.

DwtNhighlightPixmap
: Specifies the pattern and color used for highlighting gadget children.

DwtNuserData
: Specifies any user private data to be associated with the widget. The XUI Toolkit does not interpret this data.

DwtNdirectionRToL
: Specifies the direction in which the text is drawn and wraps. You can pass DwtDirectionLeftDown (text is drawn from left to right and wraps down); DwtDirectionRightUp (text is drawn from left to right and wraps up); DwtDirectionLeftDown (text is drawn from right to left and wraps down); or DwtDirectionLeftUp (text is drawn from right

to left and wraps up).

`DwtNfont` Specifies the font of the text used in gadget children.

`DwtNhelpCallback`
Specifies the callback function or functions called when a help request is made.

`DwtNunits` Specifies the type of units for the `DwtNx` and `DwtNy` attributes. You use these when adding child widgets to the dialog box. The `DwtNunits` attribute cannot be changed after the widget is created. You can pass `DwtPixelUnits` or `DwtFontUnits`.

`DwtNstyle` Specifies the style of the dialog box widget. For `DwtDialogBoxPopupCreate` you can pass `DwtModal` or `DwtModeless`. For `DwtDialogBoxCreate` you can pass `DwtWorkarea`. You cannot change `DwtNstyle` after the widget is created.

`DwtNfocusCallback`
Specifies the callback function or functions called when the dialog box accepted the input focus. For this callback, the reason is `DwtCRFocus`.

`DwtNtextMergeTranslations`
Specifies the translation manager syntax that will be merged with each text widget.

`DwtNmarginWidth` Specifies the number of pixels between the maximum right border of a child widget window and the dialog box.

`DwtNmarginHeight`
Specifies the number of pixels between the maximum bottom border of a child widget window and the dialog box.

`DwtNdefaultPosition`
Specifies a boolean value that, when `True`, causes `DwtNx` and `DwtNy` to be ignored and forces the default widget position. The default widget position is centered in the parent window. If `False`, the specified `DwtNx` and `DwtNy` attributes are used to position the widget.

If the dialog box is displayed partially off the screen as a result of being centered in the parent window, the centering rule is violated. When this occurs, the parent window is repositioned so that the entire dialog box is displayed on the screen.

The pop-up dialog box is recentered every time it is popped up. Consequently, if the parent moves in between invocations of the dialog box, the box pops up centered in the parent window's new location. However, the dialog box does not dynamically follow its parent while it is displayed. If the parent is moved, the dialog box will not move until the next time it is popped up.

If the user moves the dialog box with the window manager, the toolkit turns off `DwtNdefaultPosition`. This results in the dialog box popping up in the location specified by the user on each subsequent invocation.

`DwtNchildOverlap`

Specifies a boolean value that, when `True`, indicates that the dialog box approves geometry requests from its children that result in one child overlapping other children. If `False`, the dialog box disapproves these geometry requests.

`DwtNresize`

Specifies how the dialog box resizes when its children are managed and unmanaged and when geometry requests occur. You can pass `DwtResizeFixed`, `DwtResizeGrowOnly`, or `DwtResizeShrinkWrap`.

`DwtResizeFixed` indicates that the dialog box does not change its size when children are added or deleted, or on geometry requests from its children.

`DwtResizeGrowOnly` indicates that the dialog box always attempts to grow as necessary when children are added or deleted, or on geometry requests from its children.

`DwtResizeShrinkWrap` indicates that the dialog box always attempts to grow or shrink to fit its current set of managed children as children are added

or deleted, or on geometry requests from its children.

`DwtNgrabKeySyms` Specifies a NULL-terminated array of keysyms. The dialog box calls the Xlib function `XGrabKey` for each keysym. `XGrabKey` specifies `AnyModifier` for *modifiers*, `GrabModeAsync` for *pointer_mode*, and `GrabModeSync` for *keyboard_mode*. The dialog box uses the `XGrabKey` function in conjunction with the value of `DwtNgrabMergeTranslations` to implement moving the focus among its children in a synchronous manner. You cannot change this attribute after the widget is created.

`DwtNgrabMergeTranslations` Specifies the parsed translation syntax to merge into the dialog box syntax to handle the key events. The syntax is merged when the dialog box is first realized. Any change made to this attribute after the dialog box is realized will not have any effect.

`DwtNtitle` Specifies the compound-string label. The label is given to the window manager for the title bar if `DwtNstyle` is `DwtModeless`.

`DwtNmapCallback` Specifies the callback function or functions called when the window is about to be mapped. For this callback, the reason is `DwtCRMap`.

`DwtNunmapCallback` Specifies the callback function or functions called when the window was unmapped. For this callback, the reason is `DwtCRUnmap`.

`DwtNtakeFocus` Specifies a boolean value that, when `True`, indicates that the dialog box takes the input focus when managed.

`DwtNnoResize` Specifies a boolean value that, when `True`, indicates that a modal or modeless dialog box does not have a window manager resize button. When `False`, the dialog box has a window manager resize button.

`DwtNautoUnmanage` Specifies a boolean value that, when `True`, indicates that the dialog box unmanages itself when

any push button is activated. This attribute cannot be changed after widget creation.

`DwtNdefaultButton`
: Specifies the ID of the push button widget that is activated when the user presses the RETURN or ENTER key.

`DwtNcancelButton`
: Specifies the ID of the push button widget that is activated when the user presses the Shift and Return keys simultaneously.

`DwtNautoUnrealize`
: Specifies a boolean value that, when `False`, indicates that the dialog box creates the window(s) for itself and its children when it is first managed, and never destroys them. If `True`, the dialog box re-creates the window(s) every time it is managed, and destroys them when it is unmanaged.

  The setting of this attribute is a performance tradeoff between the client cpu load (highest when set to `True`), and the server window load (highest when set to `False`).

The following constraint attributes are passed on to any widget that is made a child of a dialog box widget. These constraint values are used only for dialog boxes that have the `DwtNunits` attribute set to `DwtFontUnits`.

`DwtNfontX`
: Specifies the placement of the left hand side of the widget window in font units. The default is the value of `DwtNx`.

`DwtNfontY`
: Specifies the placement of the top of the widget window in font units. The default is the value of `DwtNy`.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                int reason;
                XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For the callbacks associated with `DwtDialogBoxCreate`, the reason member can be set to:

`DwtCRFocus`	The dialog box has received the input focus.
`DwtCRHelpRequested`	The user has selected Help.

For the callbacks associated with `DwtDialogBoxPopupCreate`, the reason member can be set to:

`DwtCRMap`	The dialog box is about to be mapped.
`DwtCRUnmap`	The dialog box is about to be unmapped.
`DwtCRFocus`	The dialog box has received the input focus.
`DwtCRHelpRequested`	The user has selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtDisplayCSMessage (3Dwt)

Name

DwtDisplayCSMessage – Displays a compound-string message.

Syntax

```
Widget DwtDisplayCSMessage(parent_widget, name,
                           default_position, x, y,
                           style, message_vector,
                           widget_id convert_proc,
                           ok_callback, help_callback)
    Widget parent_widget;
    char *name;
    int default_position;
    int x, y;
    int style;
    int *message_vector;
    Widget *widget;
    int (*convert_proc)();
    DwtCallbackPtr ok_callback;
    DwtCallbackPtr help_callback;
```

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, indicates that `DwtNx` and `DwtNy` are to be ignored forcing the widget to be centered in the parent window.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window.

style Specifies the style of the message box widget. You can pass `DwtModal` (modal) or `DwtModeless` (modeless).

message_vector
Specifies the message argument vector identifying the

message identifier and associated information.

The first longword contains the number of longwords in the message blocks to follow. The first longword in each message block contains a pointer to the compound-string. The next word consists of the FAO parameter count. The final n longwords in the message block are the FAO parameters.

widget_id This argument contains the widget ID of an already-existing message box widget. If this argument is nonzero, a new message box is not created. An `XtSetValues` will be performed on this widget to change the text of the message to match this new message. This is an input/output argument. That is, the function fills in *widget_id* after you call it.

convert_proc Specifies a pointer to a function that is executed after the message is formatted but before it is displayed.

A pointer to the formatted compound-string is passed to the function as a parameter. This parameter is a NULL-terminated character string.

ok_callback Specifies the callback function or functions called when the user clicks on the Acknowledged push button. For this callback, the reason is `DwtCRYes`.

help_callback Specifies the callback function or functions called when a help request is made.

Description

The `DwtDisplayCSMessage` function accepts an array of compound-strings, formats them, and creates a message box.

If the function returns a zero, the message is not appended to the messages to be displayed.

Return Value

Upon completion, `DwtDisplayCSMessage` returns to the calling program the ID of the created message box widget.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtDisplayVmsMessage – Accepts and displays a VMS message.

Syntax

```
Widget DwtDisplayVmsMessage(parent_widget, name,
                            default_position, x, y,
                            style, message_vector,
                            widget_id, convert_proc,
                            ok_callback, help_callback)
    Widget parent_widget;
    char *name;
    int default_position;
    int x, y;
    int style;
    int *message_vector;
    Widget *widget_id;
    int (*convert_proc)();
    DwtCallbackPtr ok_callback;
    DwtCallbackPtr help_callback;
```

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, indicates that `DwtNx` and `DwtNy` are to be ignored forcing the widget to be centered in the parent window.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window.

style Specifies the style of the message box widget. You can pass `DwtModal` (modal) or `DwtModeless` (modeless).

message_vector
Specifies the message argument vector identifying the

message identifier and associated information. This argument is identical to the VMS $PUTMSG system service.

The first longword contains the number of longwords in the message blocks to follow. The first longword in each message block contains a pointer to the VMS message identifier. Message identifiers are passed by value.

If the message is user-supplied, the next word consists of the $FAO parameter count. The final n longwords in the message block are the $FAO parameters.

widget_id This argument contains the widget ID of an already-existing message box widget. If this argument is nonzero, a new message box is not created. An `XtSetValues` will be performed on this widget to change the text of the message to match this new message. This is an input/output argument. That is, the function fills in *widget_id* after you call it.

convert_proc Specifies a pointer to a function that is executed after the message is formatted but before it is displayed. A pointer to the formatted string is passed to the function as a parameter. This parameter is a NULL-terminated character string.

ok_callback Specifies the callback function or functions called when the user clicks on the Acknowledged push button. For this callback, the reason is `DwtCRYes`.

help_callback Specifies the callback function or functions called when a help request is made.

Description

The `DwtDisplayVmsMessage` function accepts standard VMS message vectors (as defined by the $PUTMSG system service), retrieves the messages, formats them, and creates a message box in which to display the message.

This parameter is a NULL-terminated character string.

Return Value

Upon completion, `DwtDisplayVmsMessage` returns to the calling program the ID of the created message box widget.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtDrmFreeResourceContext (3Dwt)

Name

DwtDrmFreeResourceContext – Frees a resource context.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtDrmFreeResourceContext(context_id)
    DRMResourceContextPtr context_id;
```

Arguments

context_id	Specifies the resource context to be freed.

Description

The `DwtDrmFreeResourceContext` function frees the memory buffer and the resource context.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMBadContext`	Invalid resource context.

See Also

DwtDrmGetResourceContext(3Dwt)

Name

DwtDrmGetResourceContext – Gets a resource context.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtDrmGetResourceContext(alloc_func, free_func,
                                  size, context_id_return)
    char *((*alloc_func) ());
    void (*free_func) ();
    DRMSize size;
    DRMResourceContextPtr *context_id_return;
```

Arguments

alloc_func	Specifies the function to use in allocating memory for this resource context. A NULL pointer means use the default, which is `XtMalloc`.
free_func	Specifies the function to use in freeing memory for this context. A NULL pointer means use the default, which is `XtFree`.
size	Specifies the size of the memory buffer to allocate.
context_id_return	Returns the new resource context.

Description

The `DwtDrmGetResourceContext` function allocates a new resource context and a memory buffer of the requested size. It then associates the buffer with the context.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMFailure`	The function failed.

DwtDrmGetResourceContext (3Dwt)

See Also

DwtDrmFreeResourceContext(3Dwt)

Name

DwtDrmHGetIndexedLiteral – Fetches indexed literals from a DRM hierarchy.

Syntax

Cardinal DwtDrmHGetIndexedLiteral(*hierarchy_id*, *index*, *context_id*)
DRMHierarchy *hierarchy_id*;
String *index*;
DRMResourceContextPtr *context_id*;

Arguments

hierarchy_id	Specifies the hierarchy to be searched.
index	Specifies the case-sensitive index of the desired literal.
context_id	Specifies the resource context into which the literal is read.

Description

The `DwtDrmHGetIndexedLiteral` function searches a DRM search hierarchy for a literal, given the literal's index. That is, it gets a public literal from a DRM search hierarchy. This function returns the literal as the contents of the context buffer. The group that is fetched is always `DRMgLiteral`. The literal type filter is taken from the context. If unmodified in the context obtained from `DwtDrmGetResourceContext`, there is no filtering (type = `RGMtNul`). In general, you do not need to set any of the fields in the context, except possibly type. The following buffer contents are for some common literal types obtained from a UID file. Note that in some cases the caller must cause offsets to be memory pointers.

```
DwtDrmRCType(context_id) == RGMrTypeChar8:
  DwtDrmRCBuffer(context_id) contains a null-terminated ASCII string

DwtDrmRCType(context_id) == RGMrTypeCString:
  DwtDrmRCBuffer(context_id) contains a compound-string (DwtCompString)

DwtDrmRCType(context_id) == RGMrTypeChar8Vector:
DwtDrmRCType(context_id) == RGMrTypeCStringVector:
  DwtDrmRCBuffer(context_id) contains an RGM text vector
  or stringtable (RGMTextVector).  The items in the
  text vector contain offsets into the buffer that
  locate either null-terminated ASCII strings or
  compound-strings.  You can relocate these to memory
  pointers by adding the buffer address to the offset:
```

```
item[n].textitem.pointer = item[n].textitem.offset+bufadr
```

Return Value

This function returns one of these status return constants:

DRMSuccess	The function executed successfully.
DRMNotFound	Literal not found.
DRMFailure	The function failed. Invalid resource context.

Name

DwtDrmRCBuffer – Returns a pointer to the memory buffer.

Syntax

```
#include <X11/DwtAppl.h>
char * DwtDrmRCBuffer(context_id)
    DRMResourceContextPtr context_id;
```

Arguments

context_id	Specifies the resource context to read.

Description

The `DwtDrmRCBuffer` macro returns a pointer to the memory buffer that contains the data for this resource context. No validity checking is done on the resource context; the caller must ensure that the resource context pointer is valid.

Return Value

Pointer to the memory buffer.

DwtDrmRCSetType (3Dwt)

Name

DwtDrmRCSetType – Modifies the type of a resource context.

Syntax

```
#include <X11/DwtAppl.h>
DwtDrmRCSetType(context_id, type)
    DRMResourceContextPtr context_id;
    DRMType *type;
```

Arguments

context_id	Specifies the resource context to modify.
type	Specifies the new value for the resource context type.

Description

The `DwtDrmRCSetType` function modifies the type of the specified resource context. No validity checking is done on the resource context; the caller must ensure that the resource context pointer is valid. No return code is defined.

See Also

DwtDrmRCType(3Dwt)

Name

DwtDrmRCSize – Returns the size of a resource context.

Syntax

```
#include <X11/DwtAppl.h>
DRMSize DwtDrmRCSize(context_id)
    DRMResourceContextPtr context_id;
```

Arguments

context_id	Specifies the resource context to read.

Description

The `DwtDrmRCSize` macro returns the size of the specified resource context. Note that no validity checking is done on the resource context; the caller must ensure that the context pointer is valid.

Return Value

This macro can return one of the following status return constants:

`DRMSuccess`	The function executed successfully.
`DRMSize`	The size in bytes of the resource context.
`DRMFailure`	Invalid resource context.

DwtDrmRCType (3Dwt)

Name

DwtDrmRCType – Returns the type of a resource context.

Syntax

```
#include <X11/DwtAppl.h>
DRMType DwtDrmRCType(context_id)
    DRMResourceContextPtr context_id;
```

Arguments

context_id	Specifies the resource context to read.

Description

The `DwtDrmRCType` macro returns the type of the specified resource context. Note that no validity checking is done on the resource context. The caller must ensure that the resource context pointer is valid.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMType`	The type of the resource context.
`DRMInvalid`	Invalid resource context.

See Also

DwtDrmRCSetType(3Dwt)

Name

DwtEndCopyFromClipboard – Notifies the cut and paste functions that the application has completed copying an item from the clipboard and unlocks the clipboard.

Syntax

int DwtEndCopyFromClipboard(*display*, *window*)
Display **display*;
Window *window*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

Description

The `DwtEndCopyFromClipboard` function unlocks the clipboard when the application has copied all data from the clipboard. If the application calls `DwtStartCopyFromClipboard`, it must call `DwtEndCopyFromClipboard`. These two functions lock and unlock the clipboard and allow the application to copy data from the clipboard incrementally.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtCopyFromClipboard (3Dwt), DwtStartCopyFromClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtEndCopyToClipboard – Places data in the clipboard data structure.

Syntax

int DwtEndCopyToClipboard(*display*, *window*, *item_id*)
Display **display*;
Window *window*;
long *item_id*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
item_id	Specifies the number assigned to this data item. This number was returned by a previous call to `DwtBeginCopyToClipboard`.

Description

The `DwtEndCopyToClipboard` function locks the clipboard from access by other applications, places data in the clipboard data structure, and unlocks the clipboard. Data items copied to the clipboard by `DwtCopyToClipboard` are not actually entered in the clipboard data structure until the call to `DwtEndCopyToClipboard`.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.

`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtCopyToClipboard (3Dwt), DwtBeginCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtFetchColorLiteral – Fetches a named color literal from a UID file.

Syntax

```
#include <X11/DwtAppl.h>
int DwtFetchColorLiteral(hierarchy_id, index, display, colormap_id,
pixel_return)
    DRMHierarchy hierarchy_id;
    String index;
    Display *display;
    Colormap colormap_id;
    Pixel *pixel_return;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the specified literal. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
index	Specifies the UIL name of the color literal to fetch. You must define this name in UIL as an exported value.
display	Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the `Display` structure see the Xlib function `XOpenDisplay`.
colormap_id	Specifies the ID of the color map. If NULL, the default color map is used.
pixel_return	Returns the ID of the color literal.

Description

The `DwtFetchColorLiteral` function fetches a named color literal from a UID file, and converts the color literal to a pixel color value.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMNotFound`	The color literal was not found in the UIL file.

`DRMFailure`	The function failed.

See Also

DwtFetchIconLiteral(3Dwt), DwtFetchLiteral(3Dwt)

Name

DwtFetchIconLiteral – Fetches a named icon literal from a hierarchy.

Syntax

```
#include <X11/DwtAppl.h>
int DwtFetchIconLiteral(hierarchy_id, index, screen,
                        display, fgpix, bgpix, pixmap_return)
    DRMHierarchy hierarchy_id;
    String index;
    Screen *screen;
    Display *display;
    Pixel fgpix;
    Pixel bgpix;
    Pixmap *pixmap_return;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the specified icon literal. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
index	Specifies the UIL name of the icon literal to fetch.
screen	Specifies the screen used for the pixmap. The *screen* argument specifies a pointer to the Xlib structure `Screen` which contains the information about that screen and is linked to the `Display` structure. For more information on the `Display` and `Screen` structures see the Xlib function `XOpenDisplay` and the associated screen information macros.
display	Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the `Display` structure see the Xlib function `XOpenDisplay`.
fgpix	Specifies the foreground color for the pixmap.
bgpix	Specifies the background color for the pixmap.
pixmap_return	Returns the resulting X pixmap value.

DwtFetchIconLiteral (3Dwt)

Description

The `DwtFetchIconLiteral` function fetches a named icon literal from a DRM hierarchy, and converts the icon literal to an X pixmap.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMNotFound`	The icon literal was not found in the hierarchy.
`DRMFailure`	The function failed.

See Also

DwtFetchLiteral(3Dwt), DwtFetchColorLiteral(3Dwt)

Name

DwtFetchInterfaceModule – Fetches all the widgets defined in an interface module in the UID hierarchy.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtFetchInterfaceModule(hierarchy_id, module_name,
                                 parent_widget, widget_return)
    DRMHierarchy hierarchy_id;
    char *module_name;
    Widget parent_widget;
    Widget *widget_return;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the interface definition. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
module_name	Specifies the name of the interface module, which you specified in the UIL module header. By convention, this is usually the generic name of the application.
parent_widget	Specifies the parent widget ID for the topmost widgets being fetched from the module. The topmost widgets are those that have no parents specified in the UIL module. The parent widget is usually the top-level widget returned by `XtInitialize`.
widget_return	Returns the widget ID for the last main window widget encountered in the UIL module, or NULL if no main window widget is found.

Description

The `DwtFetchInterfaceModule` function fetches all the widgets defined in a UIL module in the UID hierarchy. Typically, each application has one or more modules that define its interface. Each must be fetched in order to initialize all the widgets the application requires. Applications do not need to define all their widgets in a single module.

If the module defines a main window widget, `DwtFetchInterfaceModule` returns its widget ID. If no main window widget is contained in the module, `DwtFetchInterfaceModule` returns NULL and no widgets are realized.

The application can obtain the IDs of widgets other than the main window widget by using creation callbacks.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMFailure`	The function failed.
`DRMNotFound`	The interface module or topmost widget not found.

Name

DwtFetchLiteral – Fetches a named literal from a UID file.

Syntax

```
#include <X11/DwtAppl.h>
int DwtFetchLiteral(hierarchy_id, index, display, value_return, type_return)
    DRMHierarchy hierarchy_id;
    String index;
    Display *display;
    caddr_t *value_return;
    DRMCode *type_return;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the specified literal. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
index	Specifies the UIL name of the literal (pixmap) to fetch. You must define this name in UIL as an exported value.
display	Specifies the display used for the pixmap. The *display* argument specifies the connection to the X server. For more information on the `Display` structure see the Xlib function `XOpenDisplay`.
value_return	Returns the ID of the named literal's value.
type_return	Returns the named literal's data type.

Description

The `DwtFetchLiteral` function reads and returns the value and type of a literal (named value) that is stored as a public resource in a single UID file. This function returns a pointer to the value of the literal. For example, an integer is always returned as a pointer to an integer, and a string is always returned as a pointer to a string.

Applications should not use `DwtFetchLiteral` for fetching icon or color literals. If this is attempted, `DwtFetchLiteral` returns an error.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMWrongType`	The operation encountered an unsupported literal type.
`DRMNotFound`	The literal was not found in the UID file.
`DRMFailure`	The function failed.

See Also

DwtFetchIconLiteral(3Dwt), DwtFetchColorLiteral(3Dwt)

Name

DwtFetchSetValues – Fetches the values to be set from literals stored in UID files.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtFetchSetValues(hierarchy_id, widget, args, num_args)
    DRMHierarchy hierarchy_id;
    Widget widget;
    ArgList args;
    Cardinal num_args;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the specified literal. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
widget	Specifies the widget that is modified.
args	Specifies an argument list that identifies the widget arguments to be modified as well as the index (UIL name) of the literal that defines the value for that argument. The name part of each argument (args[n].name) must begin with the string `DwtN` followed by the name that uniquely identifies this attribute tag. For example, `DwtNwidth` is the attribute name associated with the core argument *width*. The value part (args[n].value) must be a string that gives the index (UIL name) of the literal. You must define all literals in UIL as exported values.
num_args	Specifies the number of entries in *args*.

Description

The `DwtFetchSetValues` function is similar to `XtSetValues`, except that the values to be set are defined by the UIL named values that are stored in the UID hierarchy. `DwtFetchSetValues` fetches the values to be set from literals stored in UID files.

This function sets the values on a widget, evaluating the values as public literal resource references resolvable from a UID hierarchy. Each literal is fetched from the hierarchy, and its value is modified and converted as

required. This value is then placed in the argument list and used as the actual value for an XtSetValues call. DwtFetchSetValues allows a widget to be modified after creation using UID file values exactly as is done for creation values in DwtFetchWidget.

As in DwtFetchWidget, each argument whose value can be evaluated from the UID hierarchy is set in the widget. Values that are not found or values in which conversion errors occur are not modified.

Each entry in the argument list identifies an argument to be modified in the widget. The name part identifies the tag, which begins with DwtN. The value part must be a string whose value is the index of the literal. Thus, the following code would modify the label resource of the widget to have the value of the literal accessed by the index OK_button_label in the hierarchy:

```
args[n].name = DwtNlabel;
args[n].value = "OK_button_label";
```

Return Value

This function returns one of these status return constants:

DRMSuccess	The function executed successfully.
DRMFailure	The function failed.

See Also

XtSetValues(3Dwt)

Name

DwtFetchWidget – Fetches and then creates any indexed (UIL named) application widget and its children.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtFetchWidget(hierarchy_id, index, parent_widget,
widget_return, class_return)
    DRMHierarchy hierarchy_id;
    String index;
    Widget parent_widget;
    Widget *widget_return;
    DRMType *class_return;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the interface definition. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
index	Specifies the UIL name of the widget to fetch.
parent_widget	Specifies the parent widget ID.
widget_return	Returns the widget ID of the created widget. If this is not NULL when you call `DwtFetchWidgetOverride`, DRM assumes that the widget has already been created and `DwtFetchWidgetOverride` returns `DRMFailure`.
class_return	Returns the class code identifying DRM's widget class. The widget class code for the main window widget, for example, is `DRMwcMainWindow`. Literals identifying DRM widget class codes are defined in `DRM.h`.

Description

The `DwtFetchWidget` function fetches and then creates an indexed application widget and its children. The indexed application widget is any widget that is named in UIL and that is not the child of any other widget in the UID hierarchy. In fetch operations, the fetched widget's subtree is also fetched and created. This widget must not appear as the child of a widget within its own subtree. `DwtFetchWidget` does not execute `XtManageChild` for the newly created widget.

DwtFetchWidget(3Dwt)

`DwtFetchWidget` fetches widgets where `DwtFetchInterfaceModule` is not used. `DwtFetchWidget` provides specific control over which widgets are fetched from a UIL file; `DwtFetchInterfaceModule`, on the other hand, fetches all widgets in a single call. An application can fetch any named widget in the UID hierarchy using `DwtFetchWidget`. `DwtFetchWidget` can be called at any time to fetch a widget that was not fetched at application startup. `DwtFetchWidget` determines if a widget has already been fetched by checking *widget_return* for a NULL value. Non-NULL values signify that the widget already has been fetched, and `DwtFetchWidget` fails. `DwtFetchWidget` can be used to defer fetching pop-up widgets until they are first referenced (presumably in a callback), and then used to fetch them once.

`DwtFetchWidget` can also create multiple instances of a widget (and its subtree). In this case, the UID definition functions as a template; a widget definition can be fetched any number of times. An application can use this to make multiple instances of a widget, for example, in a dialog box box or menu.

The index (UIL name) that identifies the widget must be known to the application.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMNotFound`	Widget not found in UID hierarchy.
`DRMFailure`	The function failed.

See Also

DwtFetchWidgetOverride(3Dwt)

Name

DwtFetchWidgetOverride – Fetches any indexed (UIL named) application widget. It overrides the arguments specified for this application widget in UIL.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtFetchWidgetOverride(hierarchy_id, index, parent_widget,
                                override_name, override_args,
                                override_num_args, widget_return,
class_return)
    DRMHierarchy hierarchy_id;
    String index;
    Widget parent_widget;
    String override_name;
    ArgList override_args;
    Cardinal override_num_args;
    Widget *widget_return;
    DRMType *class_return;
```

Arguments

hierarchy_id	Specifies the ID of the UID hierarchy that contains the interface definition. The *hierarchy_id* was returned in a previous call to `DwtOpenHierarchy`.
index	Specifies the UIL name of the widget to fetch.
parent_widget	Specifies the parent widget ID.
override_name	Specifies the name to override the widget name. Use a NULL value if you do not want to override the widget name.
override_args	Specifies the override argument list, exactly as would be given to `XtCreateWidget` (conversion complete and so forth). Use a NULL value if you do not want to override the argument list.
override_num_args	Specifies the number of arguments in *override_args*.
widget_return	Returns the widget ID of the created widget. If this is not NULL when you call `DwtFetchWidgetOverride`, DRM assumes that the widget has already been created and

`DwtFetchWidgetOverride` returns `DRMFailure`.

class_return Returns the class code identifying DRM's widget class. The widget class code for the main window widget, for example, is `DRMwcMainWindow`. Literals identifying DRM widget class codes are defined in `DRM.h`.

Description

The `DwtFetchWidgetOverride` function is the extended version of `DwtFetchWidget`. It is identical to `DwtFetchWidget`, except that it allows the caller to override the widget's name and any arguments that `DwtFetchWidget` would otherwise retrieve from the UID file or one of the defaulting mechanisms. That is, the override argument list is not limited to those arguments in the UID file.

The override arguments apply only to the widget fetched and returned by this function. Its children (subtree) do not receive any override parameters.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMNotFound`	Widget not found in UID hierarchy.
`DRMFailure`	The function failed.

See Also

DwtFetchWidget(3Dwt)

Name

DwtFileSelection, DwtFileSelectionCreate – Creates a file selection box widget for the application to query the user for a file selection.

Syntax

Widget DwtFileSelection(*parent_widget*, *name*, *x*, *y*,
title, *value*, *dirmask*,
visible_items_count, *style*, *default_position*,
default_position, *callback*,
help_callback)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString *title*;
DwtCompString *value*;
DwtCompString *dirmask*;
int *visible_items_count*;
int *style*;
Boolean *default_position*;
DwtCallbackPtr *callback*, *help_callback*;

Widget DwtFileSelectionCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core

widget attribute.

title Specifies the text that appears in the banner of the file selection box. This argument sets the DwtNtitle attribute associated with DwtDialogBoxPopupCreate.

value Specifies the selected file. The file name appears in the text entry field and is highlighted in the list box, if present. This argument sets the DwtNvalue attribute associated with DwtSelectionCreate.

dirmask Specifies the directory mask used in determining the files displayed in the file selection list box. This argument sets the DwtNdirMask attribute associated with DwtFileSelectionCreate.

visible_items_count
Specifies the maximum number of files visible at one time in the file selection list box. This argument sets the DwtNvisibleItemsCount attribute associated with DwtSelectionCreate.

style Specifies the style of the pop-up dialog box widget. You can pass DwtModal (modal) or DwtModeless (modeless). This argument sets the DwtNstyle attribute associated with DwtDialogBoxPopupCreate.

default_position
Specifies a boolean value that, when True, causes DwtNx and DwtNy to be ignored and forces the default widget position. The default widget position is centered in the parent window. If False, the specified DwtNx and DwtNy attributes are used to position the widget. This argument sets the DwtNdefaultPosition attribute associated with DwtDialogBoxCreate.

callback Specifies the callback function or functions called when the user makes or cancels a selection, or there is no match for the item selected by the user. This argument sets the DwtNactivateCallback, DwtNcancelCallback, and DwtNnoMatchCallback attributes associated with DwtSelectionCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtFileSelection` and `DwtFileSelectionCreate` functions create an instance of a file selection widget for the application to query the user for a file selection and return its associated widget ID. When calling `DwtFileSelection`, you set the file selection box widget attributes presented in the formal parameter list. For `DwtFileSelectionCreate`, however, you specify a list of attribute name/value pairs that represent all the possible file selection box widget attributes.

This is a subclass of the selection widget, which is a subclass of the dialog widget. The file selection widget is a specialized pop-up dialog box, supporting either modal or modeless formats.

A file selection widget contains the following:

- A list box displaying the file names from which to choose
- A directory mask text entry field
- A selection text entry field
- An Apply push button to apply the dirmask to generate a new list of files
- An Ok push button to inform the application that the user made a selection
- A Cancel push button to inform the application that the user canceled a selection

Note that the callback data structure also includes the current `DwtNvalue` and `DwtNdirMask`. This allows user input text and directory information to be passed back.

The file selection widget supports remote file search between nodes on a network. You can perform remote file searches from VMS to ULTRIX systems, but currently not from ULTRIX to VMS systems.

DwtFileSelection (3Dwt)

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Centered in the parent window
DwtNy	Position	Centered in the parent window
DwtNwidth	Dimension	The width of the list box, plus the width of the push buttons, plus three times DwtNmarginWidth. The list box will grow to accommodate items wider than the title.
DwtNheight	Dimension	The height of the list box, plus the height of the text edit field, plus the height of the label, plus three times DwtNmarginHeight.
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL
Dialog Pop-Up Attributes		
DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL

DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNunits	unsigned char	DwtFontUnits
DwtNstyle	unsigned char	DwtModal
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	XtTranslations	NULL
DwtNmarginWidth	Dimension	5 pixels
DwtNmarginHeight	Dimension	5 pixels
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeGrowOnly
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNtitle	DwtCompString	"Open"
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	Widget	NULL
DwtNcancelButton	Widget	NULL

Selection Attributes

DwtNlabel	DwtCompString	"Items"
DwtNvalue	DwtCompString	""
DwtNokLabel	DwtCompString	"Ok"
DwtNcancelLabel	DwtCompString	"Cancel"
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNcancelCallback	DwtCallbackPtr	NULL
DwtNnoMatchCallback	DwtCallbackPtr	NULL
DwtNvisibleItemsCount	int	8
DwtNitems	DwtCompString *	NULL
DwtNitemsCount	int	Zero
DwtNmustMatch	Boolean	False
DwtNselectionLabel	DwtCompString	"Files in"

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNfilterLabel	DwtCompString	"File filter"
DwtNapplyLabel	DwtCompString	"Filter"
DwtNdirMask	DwtCompString	"*.*"
DwtNdirSpec	DwtCompString	""
DwtNfileSearchProc	VoidProc	FileSelectionSearch (ULTRIX default directory file search function)
DwtNlistUpdated	Boolean	False
DwtNfileToExternProc	VoidProc	NULL
DwtNfileToInternProc	VoidProc	NULL
DwtNmaskToExternProc	VoidProc	NULL
DwtNmaskToInternProc	VoidProc	NULL

DwtNfilterLabel
: Specifies the label for the search filter located above the text-entry field.

DwtNapplyLabel
: Specifies the label for the Apply push button.

DwtNdirMask
: Specifies the directory mask used in determining the files displayed in the file selection list box.

DwtNdirSpec
: Specifies the full ULTRIX file specification. This attribute is write only and cannot be modified by XtSetValues.

DwtNfileSearchProc
: Specifies a directory search procedure to replace the default file selection search procedure. The file selection widget's default file search procedure fulfills the needs of most applications. However, it is impossible to cover the requirements of all applications; therefore, you can replace the default search procedure.

  You call the file search procedure with two arguments: the file selection widget and the DwtFileSelectionCallbackStruct structure. The callback structure contains all required information to conduct a directory search, including the current file search mask. Once called, it is up to the search routine to generate a new list of files and update the file selection widget by using

XtSetValues.

You must set these attributes: DwtNitems, DwtNitemsCount, DwtNlistUpdated, and DwtNdirSpec. Set DwtNitems to the new list of files. If there are no files, set this attribute to NULL. This argument sets the DwtNitems attribute associated with DwtSelectionCreate.

If there are no files set DwtNitemsCount to zero. This argument sets the DwtNitemsCount associated with DwtSelectionCreate. Always set DwtNlistUpdated to True when updating the file list using a search procedure, even if there are no files. Setting DwtNdirSpec is optional, but recommended. Set this attribute to the full file specification of the directory searched. The directory specification is displayed above the list box.

DwtNlistUpdated
: Specifies an attribute that is set only by the file search procedure. Set to True, if the file list has been updated.

DwtNfileToExternProc
: Converts native, internal file names to custom, external file names displayed to the user.

DwtNfileToInternProc
: Converts custom, external file names displayed to the user to native, internal file names.

DwtNmaskToExternProc
: Converts native, internal directory masks to custom, external directory masks displayed to the user.

DwtNmaskToInternProc
: Converts custom, external directory masks displayed to the user to native, internal directory masks.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
                    DwtCompString value;
                    int value_len;
                    DwtCompString dirmask;
                    int dirmask_len;
} DwtFileSelectionCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRActivate`	The user activated the Ok push button.
`DwtCRCancel`	The user activated the Cancel button.
`DwtCRHelpRequested`	The user selected help somewhere in the file selection box.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*. The value member is set to the current selection when the callback occurred. The value_len member is set to the length of the selection compound-string. The dirmask member is set to the current directory mask when the callback occurred. The dirmask_len member is set to the length of the directory mask compound-string.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtFileSelectionDoSearch – Initiates a search with a directory mask option. Otherwise, the current directory mask is used.

Syntax

void DwtFileSelectionDoSearch(*widget, dirmask*)
FileSelectionWidget *widget*;
DwtCompString *dirmask*;

Arguments

widget	Specifies the pointer to the file selection widget data structure.
dirmask	Specifies the directory mask used in determining the files displayed in the file selection list box. This is an optional attribute. If you do not specify a directory mask, the default directory mask is used. This argument sets the `DwtNdirMask` attribute associated with `DwtFileSelectionCreate`.

Description

The file selection widget initiates file searches when any of the following occur:

- The file selection widget becomes visible (managed).
- You use `XtSetValues` to change the directory mask.
- The user clicks on the Apply push button.
- The application calls `DwtFileSelectionDoSearch`, which is another way for applications to initiate a directory search. This may be useful, for example, when the application creates a new file and wants to reflect this change in a mapped file search widget.

See Also

DwtFileSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtGetNextSegment (3Dwt)

Name

DwtGetNextSegment – Gets information about the next segment in the compound-string.

Syntax

```
int DwtGetNextSegment(context, text_return,
                      charset_return, direction_r_to_l_return,
                      lang_return, rend_return)
    DwtCompStringContext *context;
    char *text_return;
    long *charset_return;
    int *direction_r_to_l_return;
    long *lang_return;
    long *rend_return;
```

Arguments

context	Specifies the context for the call to `DwtInitGetSegment`. You initialize the context by calling `DwtInitGetSegment`, and it gets incremented each time you call `DwtGetNextSegment`.
text_return	Returns the text in the next segment.
charset_return	Returns the character set in the next segment. Values for this argument can be found in the required file `/usr/include/cda_def.h`.
direction_r_to_l_return	Returns the character direction value.
lang_return	For future use.
rend_return	For future use.

Description

The `DwtGetNextSegment` function obtains information about the next segment of the compound-string as determined by the context. The space for the resulting compound-string is allocated with this function. After using this function, you should free this space by calling `XtFree`.

Return Value

This function returns one of these status return constants:

`DwtEndCS`	The context is at the end of the compound-string.
`DwtFail`	The context is not valid.
`DwtSuccess`	Normal completion.

See Also

DwtInitGetSegment (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtGetUserData (3Dwt)

Name

DwtGetUserData – Returns the user data associated with the widget.

Syntax

char * DwtGetUserData(*widget*)
Widget *widget*;

Arguments

widget	Specifies a pointer to the widget data structure.

Description

The `DwtGetUserData` function returns any private user data associated with the widget. The returned data is not interpreted by the toolkit.

Return Value

Any private user data to be associated with the widget. The data is not interpreted by the toolkit.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtHelp, DwtHelpCreate – Creates a help menu widget.

Syntax

Widget DwtHelp(*parent_widget*, *name*, *default_position*,
x, *y*, *application_name*,
library_type, *library_spec*, *first_topic*,
overview_topic, *glossary_topic*, *unmap_callback*)
Widget *parent_widget*;
DwtCompString *name*;
Boolean *default_position*;
Position *x*, *y*;
DwtCompString *application_name*;
int *library_type*;
DwtCompString *library_spec*;
DwtCompString *first_topic*;
DwtCompString *overview_topic*;
DwtCompString *glossary_topic*;
DwtCallbackPtr *unmap_callback*;

Widget DwtHelpCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, indicates that `DwtNx` and `DwtNy` will be ignored forcing the default. By default the help widget is positioned so that it does not occlude the parent widget on the screen. This argument sets the `DwtNdefaultPosition` attribute associated with `DwtHelpCreate`.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the

parent window. This argument sets the DwtNx core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNy core widget attribute.

application_name
Specifies the application name to be used in the widget title bar. This argument sets the DwtNapplicationName attribute associated with DwtHelpCreate.

library_type Specifies the type of help topic library specified by DwtNlibrarySpec. You can pass DwtTextLibrary, which is an ULTRIX help directory. This argument sets the DwtNlibraryType attribute associated with DwtHelpCreate.

library_spec Specifies a host system file specification that identifies the help topic library, for example, /usr/help/decwhelp on UNIX-based systems. This argument sets the DwtNlibrarySpec attribute associated with DwtHelpCreate.

first_topic Specifies the first help topic to be displayed. If you pass a NULL string, the help menu widget displays a list of level one topics. This argument sets the DwtNoverviewTopic attribute associated with DwtHelpCreate.

overview_topic Specifies the application overview topic. This argument sets the DwtNoverviewTopic attribute associated with DwtHelpCreate.

glossary_topic Specifies the application glossary topic. If you pass a NULL string, the Visit Glossary entry does not appear in the widget's View pull-down menu. This argument sets the DwtNglossaryTopic attribute associated with DwtHelpCreate.

unmap_callback
Specifies the callback function or functions called when the help menu widget window was unmapped. For this callback, the reason is DwtCRUnmap. This argument sets the DwtCRUnmap attribute associated with DwtHelpCreate.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtHelp` and `DwtHelpCreate` functions create an instance of a help menu widget and return its associated widget ID.

The help menu widget is a modeless widget that allows the application to display appropriate user assistance information in response to a user request. The help menu widget displays an initial help topic and then gives the user the ability to select and view additional help topics.

The `DwtNfirstTopic` attribute allows the application to provide context-sensitive help by selecting a specific topic based on implicit or explicit cues from the user.

The format of the `DwtNfirstTopic`, `DwtNoverviewTopic`, and `DwtNglossaryTopic` compound-strings depends on `DwtNlibraryType`. If `DwtNlibraryType` is `DwtTextLibrary`, the topic string is a sequence of help library keys separated by one or more spaces.

Once the widget has been created, you can change the help topic by specifying a new `DwtNfirstTopic` by calling `XtSetValues`, and then causing the help menu widget to appear by calling `XtManageChild`.

When the user terminates a help session (using the Exit function), the widget is automatically unmanaged.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager

DwtNwidth	Dimension	Cannot be set by the caller. The help menu widget calculates the width, based on the size of the text window (DwtNcols and DwtNrows).
DwtNheight	Dimension	Cannot be set by the caller. The help menu widget calculates the height, based on the size of the text window (DwtNcols and DwtNrows).
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNaboutLabel	DwtCompString	"About"
DwtNaddtopicLabel	DwtCompString	" Additional topics"
DwtNapplicationName	DwtCompString	NULL
DwtNbadframeMessage	DwtCompString	"Couldn't find frame !CS"
DwtNbadlibMessage	DwtCompString	"Couldn't open library !CS"
DwtNcacheHelpLibrary	Boolean	False
DwtNcloseLabel	DwtCompString	"Exit"
DwtNcols	int	Language-dependent. The American English default is 55.
DwtNcopyLabel	DwtCompString	"Copy"
DwtNdefaultPosition	Boolean	True
DwtNdismissLabel	DwtCompString	"Dismiss"
DwtNeditLabel	DwtCompString	"Edit"
DwtNerroropenMessage	DwtCompString	"Error opening file !CS"
DwtNexitLabel	DwtCompString	"Exit"
DwtNfileLabel	DwtCompString	"File"
DwtNfirstTopic	DwtCompString	NULL
DwtNglossaryLabel	DwtCompString	"Glossary"
DwtNglossaryTopic	DwtCompString	NULL
DwtNgobackLabel	DwtCompString	"Go Back"
DwtNgobacktopicLabel	DwtCompString	"Go Back"
DwtNgooverLabel	DwtCompString	"Go To Overview"
DwtNgotoLabel	DwtCompString	"Go To"
DwtNgototopicLabel	DwtCompString	"Go To Topic"
DwtNhelpAcknowledgeLabel	DwtCompString	"Acknowledge"
DwtNhelpFont	DwtFontList	Language-dependent. The American English default is "-*-TERMINAL-MEDIUM-R-NARROW--*-140-*-*-C-*-ISO8859-1"
DwtNhelpLabel	DwtCompString	"Using Help"
DwtNhelphelpLabel	DwtCompString	"Overview"
DwtNhelpOnHelpTitle	DwtCompString	"Using Help"
DwtNhelpontitleLabel	DwtCompString	"Help on "
DwtNhelptitleLabel	DwtCompString	"Help"
DwtNhistoryLabel	DwtCompString	"History..."
DwtNhistoryboxLabel	DwtCompString	"Search Topic History"
DwtNkeywordLabel	DwtCompString	"Keyword..."

DwtNkeywordsLabel	DwtCompString	"Keyword "
DwtNlibrarySpec	DwtCompString	NULL
DwtNlibraryType	int	DwtTextLibrary
DwtNnokeywordMessage	DwtCompString	"Couldn't find keyword !CS"
DwtNnotitleMessage	DwtCompString	"No title to match string !CS"
DwtNnulllibMessage	DwtCompString	"No library specified\n"
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNoverviewTopic	DwtCompString	NULL
DwtNrows	int	Language-dependent. The American English default is 20.
DwtNsaveasLabel	DwtCompString	"Save As..."
DwtNsearchapplyLabel	DwtCompString	"Apply"
DwtNsearchkeywordboxLabel	DwtCompString	"Search Topic Keywords"
DwtNsearchLabel	DwtCompString	"Search"
DwtNsearchtitleboxLabel	DwtCompString	"Search Topic Titles"
DwtNselectallLabel	DwtCompString	"Select All"
DwtNtitleLabel	DwtCompString	"Title..."
DwtNtitlesLabel	DwtCompString	"Title "
DwtNtopictitlesLabel	DwtCompString	"Topic Titles "
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNviewLabel	DwtCompString	"View"
DwtNvisitglosLabel	DwtCompString	"Visit Glossary"
DwtNvisitLabel	DwtCompString	"Visit"
DwtNvisittopicLabel	DwtCompString	"Visit Topic"

DwtNaboutLabel
: Specifies the text for one of the pull-down menu entries displayed when the user clicks on the Help entry on the menu bar.

DwtNaddtopicLabel
: Specifies the text for the label indicating additional topics for help.

DwtNapplicationName
: Specifies the application name to be used in the widget title bar.

DwtNbadframeMessage
: Specifies the text for the message displayed when a frame could not be found.

DwtNbadlibMessage
: Specifies the text for the message displayed when a

requested library could not be found.

`DwtNcacheHelpLibrary`
: Specifies a boolean value that, when `True`, indicates that the text is stored in cache memory. If `False`, the text is not stored in cache memory.

`DwtNcloseLabel`
: Specifies the label for the Exit push button in the help widget window.

`DwtNcols`
: Specifies the width, in characters, of the Help Menu text window.

`DwtNcopyLabel`
: Specifies the text for the copy entry on the pull-down menu under Edit on the help widget menu bar.

`DwtNdefaultPosition`
: Specifies a boolean value that, when `True`, indicates that `DwtNx` and `DwtNy` will be ignored forcing the default. By default the help widget is positioned so that it does not occlude the parent widget on the screen.

`DwtNdismissLabel`
: Specifies the text for the push button label used to dismiss a help widget dialog box (for example, Search History, Search Title, Search Keyword boxes).

`DwtNeditLabel`
: Specifies the text for the edit entry on the help window menu bar.

`DwtNerroropenMessage`
: Specifies the text for the error message displayed when a file cannot be opened.

`DwtNexitLabel`
: Specifies the text for the push button or pull-down menu entry that allows the user to exit from help.

`DwtNfileLabel`
: Specifies the text for the file entry on the help window menu bar.

`DwtNfirstTopic`
: Specifies the first help topic to be displayed. If you pass a NULL string, the help menu widget displays a list of level one topics.

`DwtNglossaryLabel`
: Specifies the text for the glossary entry on the pull-down menu under Help on a help window menu bar.

`DwtNglossaryTopic`
: Specifies the application glossary topic. If you pass a NULL string, the Visit Glossary entry does not appear in the widget's View pull-down menu.

`DwtNgobackLabel`
: Specifies the text for a label used on the pull-down menu under View. Clicking on this object returns the user to the previous topic displayed.

`DwtNgobacktopicLabel`
: Specifies the label for the Go Back push button in the help widget window.

`DwtNgooverLabel`
: Specifies the text for a label used on the pull-down menu under View. Clicking on this label causes the Overview of Help to appear in the Help window.

`DwtNgotoLabel`
: Specifies the text for the label used on a push button in the help widget's dialog boxes. Clicking on this object after selecting a new topic displays help on the new topic in the same Help window.

`DwtNgototopicLabel`
: Specifies the label for the Go To Topic menu entry in the View pull-down menu.

`DwtNhelpAcknowledgeLabel`
: Specifies the label for the Acknowledge push button in the error message box.

`DwtNhelpFont`
: Specifies the font of the text displayed in the help menu widget.

`DwtNhelphelpLabel`
: Specifies the label for the Overview menu item in the Using Help pull-down menu.

`DwtNhelpLabel`
: Specifies the text for the label on the pull-down menu under Help.

`DwtNhelpOnHelpTitle`
: Specifies the label for the title bar in the Help-on-Help help widget.

`DwtNhelpontitleLabel`
: Specifies the label for the help widget title bar used in conjunction with the application name.

DwtNhelptitleLabel
: Specifies the label for the help widget title bar when no application name is specified.

DwtNhistoryLabel
: Specifies the text for the label in the pull-down menu under Help.

DwtNhistoryboxLabel
: Specifies the text for the label used in a history box.

DwtNkeywordLabel
: Specifies the text for the label in the pull-down menu under Help.

DwtNkeywordsLabel
: Specifies the text for the label used in a Search Topic Keyword box to identify the text entry field.

DwtNlibrarySpec
: Specifies a host system file specification that identifies the help topic library, for example, `/usr/help/decwhelp` on UNIX-based systems.

DwtNlibraryType
: Specifies the type of help topic library specified by `DwtNlibrarySpec`. You can pass `DwtTextLibrary`, which is an ULTRIX help directory.

DwtNmapCallback
: Specifies the callback function or functions called when the help widget is about to be mapped.

DwtNnokeywordMessage
: Specifies the text for the message displayed when a requested keyword cannot be found.

DwtNnotitleMessage
: Specifies the text for the message displayed when a requested title cannot be found.

DwtNnulllibMessage
: Specifies the text for the message displayed when no library has been specified.

DwtNoverviewTopic
: Specifies the application overview topic.

DwtNrows
: Specifies the height, in characters, of the Help Menu text window.

`DwtNsaveasLabel` Specifies the text for an entry on a pull-down menu under File on the Help menu bar. Clicking on this entry allows a user to save the current help text in a file. A file selection dialog box is displayed.

`DwtNsearchapplyLabel`
Specifies the text for the push button label used to initiate a search action in a Search dialog box.

`DwtNsearchkeywordboxLabel`
Specifies the text for the label used in a Search Topic Keywords box.

`DwtNsearchLabel` Specifies the text for an entry on a Help window menu bar.

`DwtNsearchtitleboxLabel`
Specifies the text for the title of a Search Topic Titles box.

`DwtNselectallLabel`
Specifies the text for an entry on the pull-down menu under Edit. Clicking on this entry selects all the text in the work area (text widget only).

`DwtNtitleLabel` Specifies the text for an entry on the pull-down menu under Search. Clicking on this entry allows a user to search for a topic by title.

`DwtNtitlesLabel` Specifies the text for the label that identifies the text entry field on the Search Topic Titles box.

`DwtNtopictitlesLabel`
Specifies the text for the label that identifies the topics found as a result of a title search in a Search Topic Titles box.

`DwtNviewLabel` Specifies the text for the View entry on a help menu bar.

`DwtNvisitglosLabel`
Specifies the text for the pull-down menu entry under View. Clicking on this entry causes the glossary to be displayed in a new Help window.

DwtNvisitLabel — Specifies the text for an entry on a push button in a help widget's dialog boxes. Clicking on this object causes information on a new topic to be displayed in a new window.

DwtNvisittopicLabel — Specifies the label for the Visit Topic menu entry in the View pull-down menu.

DwtNunmapCallback — Specifies the callback function or functions called when the help menu widget window was unmapped. For this callback, the reason is DwtCRUnmap.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRUnmap — The help window was unmapped.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtInitGetSegment (3Dwt)

Name

DwtInitGetSegment – Returns the initialized context of the compound-string.

Syntax

int DwtInitGetSegment(*context*, *compound_string*)
DwtCompStringContext **context*;
DwtCompString *compound_string*;

Arguments

context Specifies a context to be filled by this function. You should have previously allocated this context.

compound_string
Specifies the compound-string.

Description

The `DwtInitGetSegment` function returns the initialized *context* associated with the compound-string you specified (*compound_string*). You must use this returned context in a call to `DwtGetNextSegment`.

Note that the performance of `DwtInitGetSegment` (used in conjunction with `DwtGetNextSegment` to fetch multiple segments from a compound-string) has degraded from Version 1.0 of the toolkit.

A new function, `DwtStringInitContext`, not only provides better performance, it also creates the context structure that you must allocate separately when using `DwtInitGetSegment`. To improve performance, convert calls from `DwtInitGetSegment` to `DwtStringInitContext`, and use `DwtStringFreeContext` to free the context structure when you are finished with it.

Return Value

This function returns one of these status return constants:

`DwtSuccess`	Normal completion.
`DwtEndCS`	The string specified in *compound_string* is NULL.
`DwtFail`	The string specified in *compound_string* is not a compound-string.

See Also

DwtGetNextSegment (3Dwt), DwtStringFreeContext (3Dwt), DwtStringInitContext (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtInitializeDRM (3Dwt)

Name

DwtInitializeDRM – Prepares an application to use DRM widget-fetching facilities.

Syntax

void DwtInitializeDRM()

Description

The `DwtInitializeDRM` function must be called to prepare an application to use DRM widget-fetching facilities. You must call this function prior to fetching a widget. However, it is good programming practice to call `DwtInitializeDRM` prior to performing any DRM operations.

`DwtInitializeDRM` initializes the internal data structures that DRM needs to successfully perform type conversion on arguments and to successfully access widget creation facilities. An application must call `DwtInitializeDRM` before it uses other DRM functions. `DwtInitializeDRM` can be called more than once. All calls after the first have no effect.

Name

DwtInquireNextPasteCount – Returns the number of data item formats available for the next paste item in the clipboard.

Syntax

int DwtInquireNextPasteCount(*display*, *window*, *count*,
max_format_name_length)
Display **display*;
Window *window*;
int **count*;
int **max_format_name_length*;

Arguments

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

count Returns the number of data item formats available for the next-paste item in the clipboard. If no formats are available, this argument equals zero. The count includes the formats that were passed by name.

max_format_name_length
Specifies the maximum length of all format names for the next-paste item in the clipboard.

Description

The `DwtInquireNextPasteCount` function returns the number of data item formats available for the next-paste item in the clipboard. This function also returns the maximum name length for all formats in which the next-paste item is stored.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.
`ClipboardNoData`	Information could not be obtained from an application using the ICCCM clipboard selection mechanism. This return value indicates that the data was not available in the requested format.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtInquireNextPasteFormat – Returns a specified format name for the next-paste item in the clipboard.

Syntax

int DwtInquireNextPasteFormat(*display*, *window*,
number, *format_name_buf*,
buffer_len, *copied_len*)
Display **display*;
Window *window*;
int *number*;
char **format_name_buf*;
unsigned long *buffer_len*;
unsigned long **copied_len*;

Arguments

display Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.

window Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

number Specifies the number of format names to be obtained. If this number *n* is greater than the number of formats for the data item, `DwtInquireNextPasteFormat` returns a zero in the *copied_len* argument.

format_name_buf
Specifies the buffer that receives the format name.

buffer_len Specifies the number of bytes in the format name buffer.

copied_len Specifies the number of bytes in the string copied to the buffer. If this argument equals zero, there is no *nth* format for the next-paste item.

DwtInquireNextPasteFormat (3Dwt)

Description

The `DwtInquireNextPasteFormat` function returns a specified format name for the next-paste item in the clipboard. If the name must be truncated, the function returns a warning status.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.
`ClipboardTruncate`	The data returned is truncated because the user did not provide a buffer that was large enough to hold the data.
`ClipboardNoData`	Information could not be obtained from an application using the ICCCM clipboard selection mechanism. This return value indicates that the data was not available in the requested format.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtInquireNextPasteLength – Returns the length of the data stored under a specified format name for the next-paste item in the clipboard.

Syntax

int DwtInquireNextPasteLength(*display*, *window*,
format_name, *length*)
Display **display*;
Window *window*;
char **format_name*;
unsigned long **length*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
format_name	Specifies the name of the format for the next-paste item.
length	Specifies the length of the next data item in the specified format. This argument equals zero if no data is found for the specified format, or if there is no item on the clipboard.

Description

The `DwtInquireNextPasteLength` function returns the length of the data stored under a specified format name for the next paste clipboard data item.

If no data is found for the specified format, or if there is no item on the clipboard, `DwtInquireNextPasteLength` returns a value of zero.

NOTE

Any format passed by name is assumed to have the *length* passed in a call to `DwtCopyToClipboard`, even though the data has not yet been transferred to the clipboard in that format.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.
`ClipboardNoData`	Information could not be obtained from an application using the ICCCM clipboard selection mechanism. This return value indicates that the data was not available in the requested format.

See Also

DwtCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtLabel, DwtLabelCreate – Creates a label widget for the application to display identification information (label) on the screen.

Syntax

Widget DwtLabel(*parent_widget*, *name*, *x*, *y*, *label*, *help_callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString *label*;
DwtCallbackPtr *help_callback*;

Widget DwtLabelCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.

label Specifies the label for the text style. This argument sets the `DwtNlabel` attribute associated with `DwtLabelCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtLabel` and `DwtLabelCreate` functions create an instance of a label widget and return its associated widget ID. When calling `DwtLabel`, you set the label widget attributes presented in the formal parameter list. For `DwtLabelCreate`, however, you specify a list of attribute name/value pairs that represent all the possible label widget attributes.

The application uses the label widget to display read only information (label) anywhere within the parent widget window. It has no standard callback other than `DwtNhelpCallback`.

Because a label widget does not support children, it always refuses geometry requests. The label widget does nothing on a resize by its parents.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	The width of the label or pixmap, plus two times `DwtNmarginWidth`
`DwtNheight`	`Dimension`	The height of the label or pixmap, plus two times `DwtNmarginHeight`
`DwtNborderWidth`	`Dimension`	zero pixels
`DwtNborder`	`Pixel`	Default foreground color
`DwtNborderPixmap`	`Pixmap`	NULL
`DwtNbackground`	`Pixel`	Default background color
`DwtNbackgroundPixmap`	`Pixmap`	NULL
`DwtNcolormap`	`Colormap`	Default color map
`DwtNsensitive`	`Boolean`	`True`

DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabelType	unsigned char	DwtCString
DwtNlabel	DwtCompString	Widget name
DwtNmarginWidth	Dimension	Two pixels for text Zero pixels for pixmap
DwtNmarginHeight	Dimension	Two pixels for text Zero pixels for pixmap
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNpixmap	Pixmap	NULL
DwtNmarginLeft	Dimension	Zero
DwtNmarginRight	Dimension	Zero
DwtNmarginTop	Dimension	Zero
DwtNmarginBottom	Dimension	Zero
DwtNconformToText	Boolean	True, if the widget is created with a width and height of zero

	`False`, if the widget is created with a non-zero width and height
`DwtNlabelType`	Specifies the label type. You can pass `DwtCString` (compound string) or `DwtPixmap` (icon data in pixmap).
`DwtNlabel`	Specifies the label for the text style.
`DwtNmarginWidth`	Specifies the number of pixels between the border of the widget window and the label.
`DwtNmarginHeight`	Specifies the number of pixels between the border of the widget window and the label.
`DwtNalignment`	Specifies the label alignment for text style. You can pass `DwtAlignmentCenter` (center alignment), `DwtAlignmentBeginning` (alignment at the beginning), or `DwtAlignmentEnd` (alignment at the end).
`DwtNpixmap`	Supplies icon data for the label. Pixmap is used when `DwtNlabelType` is defined as `DwtNpixmap`.
`DwtNmarginLeft`	Specifies the number of pixels that are to remain inside the left margin (`DwtNmarginWidth`) of the widget before the label is drawn.
`DwtNmarginRight`	Specifies the number of pixels that are to remain inside the right margin (`DwtNmarginWidth`) of the widget before the label is drawn.
`DwtNmarginTop`	Specifies the number of pixels that are to remain inside the top margin (`DwtNmarginTop`) of the widget before the label is drawn.
`DwtNmarginBottom`	Specifies the number of pixels that are to remain inside the bottom margin (`DwtNmarginTop`) of the widget before the label is drawn.
`DwtNconformToText`	Specifies a boolean value that indicates whether or not the widget always attempts to be just big enough to contain the label. If `True`, an `XtSetValues`

with a new label string causes the widget to attempt to shrink or expand to fit exactly (accounting for margins) the new label string. Note that the results of the attempted resize are up to the geometry manager involved. If `False`, the widget never attempts to change size on its own.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                int reason;
                XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRHelpRequested` The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtLabelGadgetCreate (3Dwt)

Name

DwtLabelGadgetCreate – Creates a label gadget.

Syntax

Widget DwtLabelGadgetCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtLabelGadgetCreate` function creates an instance of the label gadget and returns its associated gadget ID. A label gadget is similar in appearance and semantics to a label widget. Like all gadgets, the label gadget does not have a window but uses the window of the closest antecedent widget. Thus, the antecedent widget provides all event dispatching for the gadget. This currently restricts gadgets to being descendents of menu or dialog class (or subclass) widgets. Drawing information such as font and color are also those of the closest antecedent widget.

Inherited Attributes

Attribute Name	Data Type	Default
Rectangle Attributes		
`DwtNx`	`Position`	Determined by the geometry manager

DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	The width of the label plus margins
DwtNheight	Dimension	The height of the label plus margins
DwtNborderWidth	Dimension	zero pixels
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNdirectionRToL	Boolean	False
DwtNhelpCallback	DwtCallbackPtr	NULL

DwtNlabel
Specifies the label for the text style.

DwtNalignment
Specifies the label alignment for text style. You can pass DwtAlignmentCenter (center alignment), DwtAlignmentBeginning (alignment at the beginning), or DwtAlignmentEnd (alignment at the end).

DwtNdirectionRToL
Specifies a boolean value that, when False, indicates that the text is drawn from left to right. If True, the text is drawn from right to left.

DwtNhelpCallback
Specifies the callback function or functions called when a help request is made.

Return Value

This function returns the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                int reason;
                XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRHelpRequested` The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtLatin1String – Creates a compound-string for the LATIN1 character set.

Syntax

DwtCompString DwtLatin1String(*text*)
char **text*;

Arguments

text	Specifies the text string to be converted to a compound-string.

Description

The `DwtLatin1String` function creates a compound-string and is provided for those application programmers who do not need to mix compound-strings containing different character sets and directions. `DwtLatin1String` assumes the character encoding of the text to be ISO_LATIN1 and the writing direction to be from left to right.

Return Value

This function returns the resulting compound-string. It has the following default values:

- For *charset* the default is `CDA$K_ISO_LATIN1`.
- For *direction_r_to_l* the default is `False` (text is drawn from left to right).
- For *language* the default is `DwtLanguageNotSpecified`.
- For *rend* the default is `DwtRendMaskNone`.

See Also

DwtCSString (3Dwt), DwtString (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBox (3Dwt)

Name

DwtListBox, DwtListBoxCreate – Creates a list box widget for the application to display large numbers of item choices or entries in a list format.

Syntax

Widget DwtListBox(*parent_widget*, *name*, *x*, *y*,
items, *item_count*, *visible_items_count*,
callback, *help_callback*, *resize*, *horiz*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString **items*;
int *item_count*, *visible_items_count*;
DwtCallbackPtr *callback*, *help_callback*;
Boolean *resize*;
Boolean *horiz*;

Widget DwtListBoxCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
items	Specifies the list of items to be displayed by the list box widget. The list of items must be unique. This argument

sets the DwtNitems attribute associated with DwtListBoxCreate.

item_count Specifies the total number of items in the list. This argument sets the DwtNitemsCount associated with DwtListBoxCreate.

visible_items_count
Specifies the maximum number of visible items contained in the list box. For example, if DwtNitemsCount is 20, but DwtNvisibleItemsCount is 5, only 5 items are visible at any one time. This argument sets the DwtNvisibleItemsCount attribute associated with DwtListBoxCreate.

callback Specifies the callback function or functions called when single callback, single confirm callback, extend callback, and extend confirm callback functions are activated. This argument sets the DwtNsingleCallback, DwtNsingleConfirmCallback, DwtNextendCallback, and DwtNextendConfirmCallback attributes associated with DwtListBoxCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

resize Specifies a boolean value that, when True, indicates the list box increases its width to accommodate items too wide to fit inside the box. If False, the width remains constant unless the caller changes the width by calling XtSetValues. If you set DwtNresize to False, it is recommended that you set DwtNhorizontal to True. This argument sets the DwtNresize attribute associated with DwtListBoxCreate.

horiz Specifies a boolean value that, when True, indicates the list box contains a horizontal scroll bar. If False, the list box does not contain a horizontal scroll bar. A horizontal scroll bar cannot be deleted or added to a list box after the list box is created. This argument sets the DwtNscrollHorizontal attribute associated with DwtListBoxCreate.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtListBox` and `DwtListBoxCreate` functions create an instance of a list box widget and return its associated widget ID. The list box widget is a composite widget that consists of a list box, a menu with gadgets, and scroll bars.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	Set as large as necessary to hold the longest item without exceeding the size of its parent
`DwtNheight`	`Dimension`	Set as large as necessary to hold the number of items specified by `DwtNvisibleItemsCount`, without exceeding the size of the parent widget
`DwtNborderWidth`	`Dimension`	One pixel
`DwtNborder`	`Pixel`	Default foreground color
`DwtNborderPixmap`	`Pixmap`	NULL
`DwtNbackground`	`Pixel`	Default background color
`DwtNbackgroundPixmap`	`Pixmap`	NULL
`DwtNcolormap`	`Colormap`	Default color map
`DwtNsensitive`	`Boolean`	`True`

DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	NOT SUPPORTED	

Scroll Window Attributes

DwtNhorizontalScrollBar	Widget	NULL
DwtNverticalScrollBar	Widget	NULL
DwtNworkWindow	Widget	NULL
DwtNshownValueAutomaticHoriz	Boolean	True
DwtNshownValueAutomaticVert	Boolean	False

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNmarginWidth	Dimension	10 pixels
DwtNmarginHeight	Dimension	4 pixels
DwtNspacing	Dimension	1 pixel
DwtNitems	DwtCompString *	NULL
DwtNitemsCount	int	Zero
DwtNselectedItems	DwtCompString *	NULL
DwtNselectedItemsCount	int	Zero

DwtNvisibleItemsCount	int	As many items as can fit in the core attribute DwtNheight. The minimum is 1.
DwtNsingleSelection	Boolean	True
DwtNresize	Boolean	True
DwtNhorizontal	Boolean	False
DwtNsingleCallback	DwtCallbackPtr	NULL
DwtNsingleConfirmCallback	DwtCallbackPtr	NULL
DwtNextendCallback	DwtCallbackPtr	NULL
DwtNextendConfirmCallback	DwtCallbackPtr	NULL

DwtNmarginWidth
: Specifies the number of pixels between the border of the widget window and the items. This attribute sets the list box menu margin width.

DwtNmarginHeight
: Specifies the number of pixels between characters of each pair of consecutive items. This attribute sets the list box menu margin height.

DwtNspacing
: Specifies in pixels the spacing between list box entries.

DwtNitems
: Specifies the list of items to be displayed by the list box widget. The list of items must be unique. When modifying DwtNitems, always update DwtNitemsCount and DwtNselectedItemsCount. When DwtNitems is NULL, DwtNitemsCount and DwtNselectedItemsCount must be zero.

DwtNitemsCount
: Specifies the total number of items in the list. When DwtNitemsCount is zero, DwtNitems does not have to be NULL. The list box widget uses DwtNitemsCount and DwtNselectedItemsCount, not DwtNitems, to determine if the list contains any items. Therefore, you must specify DwtNitemsCount whenever you modify DwtNitems.

DwtNselectedItems
: Specifies the list of items that are selected in the list box. The last selected item is visible in the list box.

DwtNselectedItemsCount

Specifies the number of items selected in the list box. When DwtNselectedItemsCount is zero, DwtNselectedItems does not have to be NULL. The list box uses DwtNselectedItemsCount not DwtNselectedItems to determine if the list contains any selected items. Therefore, you must specify DwtNselectedItemsCount whenever you modify DwtNselectedItems.

DwtNvisibleItemsCount

Specifies the maximum number of visible items contained in the list box. For example, if DwtNitemsCount is 20, but DwtNvisibleItemsCount is 5, only 5 items are visible at any one time.

The list box widget is designed so that its height is based on DwtNvisibleItemsCount. Therefore, it is preferable to control the list box height by using DwtNvisibleItemsCount rather than DwtNheight.

Applications that control list box height through the core attribute DwtNheight are responsible for handling font changes.

DwtNsingleSelection

Specifies a boolean value that, when True, indicates only one item can be selected at a time.

DwtNresize

Specifies a boolean value that, when True, indicates the list box increases its width to accommodate items too wide to fit inside the box. If False, the width remains constant unless the caller changes the width by calling XtSetValues. If you set DwtNresize to False, it is recommended that you set DwtNhorizontal to True.

DwtNhorizontal

Specifies a boolean value that, when True, indicates the list box contains a horizontal scroll bar. If False, the list box does not contain a horizontal scroll bar. A horizontal scroll bar cannot be deleted or added to a list box after the list box is created.

DwtNsingleCallback

Specifies the callback function or functions called when the user selects a single item by clicking MB1 on a single item. For this callback, the reason is `DwtCRSingle`.

DwtNsingleConfirmCallback
: Specifies the callback function or functions called when the user double clicked MB1 on an item. For this callback, the reason is DwtCRSingleConfirm.

DwtNextendCallback
: Specifies the callback function or functions called when the user single clicks MB1 while depressing the Shift key when more than one item is selected (multiple selection callback). See the DwtNsingleSelection attribute. For this callback, the reason is DwtCRExtend.

DwtNextendConfirmCallback
: Specifies the callback function or functions called when the user double clicks MB1 while depressing the Shift key when more than one item is selected (multiple selection callback). See the DwtNsingleSelection attribute. For this callback, the reason is DwtCRExtend.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        DwtCompString item;
                        int item_length;
                        int item_number;
} DwtListBoxCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRSingle
: The user selected a single item in the list by clicking MB1 on the item.

DwtCRSingleConfirm	The user selected a single item in the list and confirmed another action to be taken (by a callback) by double clicking on an item. For example, a double click on a file in the file selection box selects that file and confirms another action to be taken.
DwtCRExtend	The user selected an item (by clicking MB1 on a single item while depressing the shift key) while there is at least one other selected item. The user clicked MB1 once while pressing the Shift key on an item when more than one is selected (multiple selection callback).
DwtCRExtendConfirm	The user selected an item and confirmed another action to be taken (by double clicking MB1 on a single item while depressing the Shift key) while there is at least one other selected item. This reason applies only if DwtNsingleSelection is True.
DwtCRHelpRequested	The user selected Help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*.

The item member is set to the last item selected when the callback occurred. Note that only the last item, not all selected items, is returned. The item_length member is set to the selected item's length when the callback occurred. The item_number member is set to the item's position in the list box when the callback occurred. The first position is one, not zero.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListBoxAddItem – Adds an item to the list within a list box widget.

Syntax

void DwtListBoxAddItem(*widget*, *item*, *position*)
Widget *widget*;
DwtCompString *item*;
int *position*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to add an item.
item	Specifies the text of the item to be added to the list box.
position	Specifies the placement of the item within the list in terms of its cell position. It uses an insert mode/cell number scheme with a 1 specifying the topmost entry position and a 0 specifying the bottom entry for adding an item to the bottom of the list.

Description

The `DwtListBoxAddItem` function adds an item to a list within the list box widget.

See Also

DwtListBoxDeleteItem (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBoxDeleteItem (3Dwt)

Name

DwtListBoxDeleteItem – Deletes an item from the list within a list box widget.

Syntax

void DwtListBoxDeleteItem(*widget*, *item*)
Widget *widget*;
DwtCompString *item*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to delete an item.
item	Specifies the text of the item to be deleted from the list box.

Description

The `DwtListBoxDeleteItem` function deletes an item from a list within the list box widget. The function searches the list for the item, removes it, and moves any subsequent entries up one cell position throughout the remaining list.

See Also

DwtListBoxAddItem (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListBoxDeletePos – Deletes an item identified by its position from the list within a list box widget.

Syntax

void DwtListBoxDeletePos(*widget*, *position*)
Widget *widget*;
int *position*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to delete an item identified by its position.
position	Specifies the position of the item to be deleted from the list.

Description

The `DwtListBoxDeletePos` function deletes an item from a list within the list box widget. The item to be deleted is identified by its position in the list. The function searches the list for the specified position, removes the item in that position, and moves any subsequent entries up one cell position throughout the remaining list.

See Also

DwtListBoxDeleteItem (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBoxDeselectAllItems (3Dwt)

Name

DwtListBoxDeselectAllItems – Deselects all of the previously selected items in a list box.

Syntax

void DwtListBoxDeselectAllItems(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to delete all previously selected items.

Description

The `DwtListBoxDeselectAllItems` function deselects (removes highlighting) all items previously selected, and removes them from the list of selected items.

See Also

DwtListBoxDeselectItem (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListBoxDeselectItem – Deselects a previously selected item in a list box.

Syntax

void DwtListBoxDeselectItem(*widget*, *item*)
Widget *widget*;
DwtCompString *item*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to delete a single previously selected item.
item	Specifies the item in the list box to be deselected (highlighting removed).

Description

The `DwtListBoxDeselectItem` function deselects (removes highlighting) an item previously selected, and removes it from the list of selected items.

See Also

DwtListBoxDeselectAllItems (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBoxDeselectPos (3Dwt)

Name

DwtListBoxDeselectPos – Deselects an item identified by its position in the list box.

Syntax

void DwtListBoxDeselectPos(*widget*, *position*)
Widget *widget*;
int *position*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to deselect an item.
position	Specifies an integer that identifies the position of the item to be deselected in the list box.

Description

The `DwtListBoxDeselectPos` function deselects an item (removes highlighting) based on its position in a list box and removes the item from the selected list.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListBoxItemExists – Verifies the existence of a particular item in a list box.

Syntax

int DwtListBoxItemExists(*widget*, *item*)
Widget *widget*;
DwtCompString *item*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to verify the existence of a specified item.
item	Specifies the item in the list box that is being searched for.

Description

The `DwtListBoxItemExists` function searches through a list box to determine if an item exists. If the specified item is found, `DwtListBoxItemExists` returns an integer that gives the position of the item in the list box. If the item is not found, `DwtListBoxItemExists` returns a zero.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBoxSelectItem (3Dwt)

Name

DwtListBoxSelectItem – Selects an item in the list box.

Syntax

void DwtListBoxSelectItem(*widget*, *item*, *notify*)
Widget *widget*;
DwtCompString *item*;
Boolean *notify*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to select an item.
item	Specifies the text of the item to be added to the list box.
notify	Specifies a boolean value that, when `True`, indicates use of this widget results in a callback to the application.

Description

The `DwtListBoxSelectItem` function selects an item in a list box, adds it to a selected item list, and calls back to the application if *notify* is `True`.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListBoxSelectPos – Selects an item identified by its position in the list box.

Syntax

void DwtListBoxSelectPos(*widget*, *position*, *notify*)
Widget *widget*;
int *position*;
Boolean *notify*;

Arguments

widget	Specifies the ID of the list box widget from whose list you want to select an item.
position	Specifies an integer that identifies the position of the item to be selected in the list box.
notify	Specifies a boolean value that, when `True`, indicates use of this widget results in a callback to the application.

Description

The `DwtListBoxSelectPos` function selects an item in a list box based on its position in the list, adds it to a selected item list, and calls back to the application, if *notify* is `True`.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBoxSetHorizPos (3Dwt)

Name

DwtListBoxSetHorizPos – Sets the horizontal position to a specified position.

Syntax

void DwtListBoxSetHorizPos(*widget*, *position*)
Widget *widget*;
int *position*;

Arguments

widget	Specifies the ID of the list box widget whose horizontal scroll bar position you want to set.
position	Specifies the position of the horizontal scroll bar in the list box widget.

Description

The `DwtListBoxSetHorizPos` function is used only if the list box has a horizontal scroll bar and the list box contains items too wide to be visible within the current list box width.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListBoxSetItem – Makes a specified item (if it exists) the first visible item in a list box, or as close to the top as possible. The item always becomes visible.

Syntax

void DwtListBoxSetItem(*widget*, *item*)
Widget *widget*;
DwtCompString *item*;

Arguments

widget	Specifies the widget ID.
item	Specifies the item to be made the first item in the list box.

Description

The `DwtListBoxSetItem` function makes the specified item (if it exists) the first visible item in a list box. The function determines which item in the list box is displayed at the top of the list box, the choice of which is limited by the `DwtNitemsCount` and `DwtNvisibleItemsCount` attributes to the list box widget. When `DwtNvisibleItemsCount` is greater than 1 and less than `DwtNitemsCount`, the list box widget fills the list box with the maximum visible items regardless of the position value.

For example, if `DwtNitemsCount` is 10 and `DwtNvisibleItemsCount` is 5, you cannot make item 8 display at the top of the list box. Instead, items 6 through 10 would be displayed. Setting *item* to the fourth item in the list would make items 4 through 8 display. If `DwtNvisibleItemsCount` is 1, you can make any item in the list be displayed at the top of the list box.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtListBoxSetPos (3Dwt)

Name

DwtListBoxSetPos – Makes a specified position (item number in the list) the top visible position in a list box, or as close to the top as possible.

Syntax

void DwtListBoxSetPos(*widget*, *position*)
Widget *widget*;
int *position*;

Arguments

widget	Specifies the ID of the list box widget whose specified item number in the list you want displayed in the top position.
position	Specifies the item number in the list displayed in the top position in the list box.

Description

The `DwtListBoxSetPos` function makes the specified position (the item number in the list) the top visible position in a list box. The function determines which item in the list box is displayed at the top of the list box, the choice of which is limited by the `DwtNitemsCount` and `DwtNvisibleItemsCount` attributes to the list box widget. When `DwtNvisibleItemsCount` is greater than 1 and less than `DwtNitemsCount`, the list box widget fills the list box with the maximum visible items regardless of the *position* value.

For example, if `DwtNitemsCount` is 10 and `DwtNvisibleItemsCount` is 5, you cannot make item 8 be displayed at the top of the list box. Instead, items 6 through 10 would be displayed. Setting *position* to 4 would make items 4 through 8 be displayed. If `DwtNvisibleItemsCount` is 1, you can make any item in the list be displayed at the top of the list box.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtListPendingItems – Returns a list of data ID/private ID pairs for a specified format name.

Syntax

int DwtListPendingItems(*display*, *window*, *format_name*,
item_list, *count*)
Display **display*;
Window *window*;
char **format_name*;
DwtClipboardPendingList **item_list*;
unsigned long **count*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
format_name	Specifies a string that contains the name of the format for which the list of data ID/private ID pairs is to be obtained.
item_list	Specifies the address of the array of data ID/private ID pairs for the specified format name. This argument is a type `DwtClipboardPendingList`. The application is responsible for freeing the memory provided by this function for storing the list.
item_count	Specifies the number of items returned in the list. If there is no data for the specified format name, or if there is no item on the clipboard, this argument equals zero.

Description

The `DwtListPendingItems` function returns a list of data ID/private ID pairs for a specified format name. For the purposes of this function, a data item is considered pending if the application originally passed it by name, the

application has not yet copied the data, and the item has not been deleted from the clipboard.

The application is responsible for freeing the memory provided by this function to store the list.

This function is used by an application when exiting to determine if the data that it passed by name should be sent to the clipboard.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtMainSetAreas – Sets up or adds the menu bar, command window, work window, and scroll bar widgets to the main window widget of the application.

Syntax

void DwtMainSetAreas(*widget*, *menu_bar*, *work_window*, *command_window*, *horizontal_scroll_bar*, *vertical_scroll_bar*)
Widget *widget*;
Widget *menu_bar*;
Widget *work_window*, *command_window*;
Widget *horizontal_scroll_bar*, *vertical_scroll_bar*;

Arguments

widget Specifies the main window widget ID.

menu_bar Specifies the widget ID for the menu bar to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNmenuBar`.

work_window Specifies the widget ID for the work window to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNworkWindow`.

command_window
Specifies the widget ID for the command window to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNcommandWindow`.

horizontal_scroll_bar
Specifies the scroll bar widget ID for the horizontal scroll bar to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNhorizontalScrollBar`.

vertical_scroll_bar
: Specifies the scroll bar widget ID for the vertical scroll bar to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNverticalScrollBar`.

Description

The `DwtMainSetAreas` function sets up or adds the menu bar, work window, command window, and scroll bar widgets to the application's main window widget. You must set these areas up before the main window widget is realized, that is, before calling the X intrinsics function `XtRealizeWidget`.

Each area is optional; therefore, you can pass NULL to one or more of these arguments. The title bar is provided by the window manager.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtMainWindow, DwtMainWindowCreate – Creates the main window widget.

Syntax

Widget DwtMainWindow(*parent_widget*, *name*, *x*, *y*, *width*, *height*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
Dimension *width*, *height*;

Widget DwtMainWindowCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID. For some applications, the parent widget ID for the main window widget is the ID returned by `XtInitialize`. However, the main window widget is not restricted to this type of parent.

name Specifies the name of the created widget.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.

width Specifies in pixels the width of the widget window. This argument sets the `DwtNwidth` core widget attribute.

height Specifies in pixels the height of the widget window. This argument sets the `DwtNheight` core widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtMainWindow` and `DwtMainWindowCreate` functions create an instance of the main window widget and return its associated widget ID. When calling `DwtMainWindow`, you set the main window widget attributes presented in the formal parameter list. For `DwtMainWindowCreate`, however, you specify a list of attribute name/value pairs that represent all the possible attributes of the main window widget.

The main window widget can contain a menu bar region, a work area with optional scroll bars, and a command area.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	5 pixels
`DwtNheight`	`Dimension`	5 pixels
`DwtNborderWidth`	`Dimension`	One pixel
`DwtNborder`	`Pixel`	Default foreground color
`DwtNborderPixmap`	`Pixmap`	NULL
`DwtNbackground`	`Pixel`	Default background color
`DwtNbackgroundPixmap`	`Pixmap`	NULL
`DwtNcolormap`	`Colormap`	Default color map
`DwtNsensitive`	`Boolean`	`True`
`DwtNancestorSensitive`	`Boolean`	The bitwise AND of the parent widget's `DwtNsensitive` and `DwtNancestorSensitive` attributes
`DwtNaccelerators`	`XtTranslations`	NULL
`DwtNdepth`	int	Depth of the parent window
`DwtNtranslations`	`XtTranslations`	NULL
`DwtNmappedWhenManaged`	`Boolean`	`True`

DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	NOT SUPPORTED	
DwtNhighlightPixmap	NOT SUPPORTED	
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNcommandWindow	Widget	NULL
DwtNworkWindow	Widget	NULL
DwtNmenuBar	Widget	NULL
DwtNhorizontalScrollBar	Widget	NULL
DwtNverticalScrollBar	Widget	NULL
DwtNacceptFocus	Boolean	False
DwtNfocusCallback	DwtCallbackPtr	NULL

DwtNcommandWindow
: Specifies the widget ID for the command window to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget.

DwtNworkWindow
: Specifies the widget ID for the work window to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget.

DwtNmenuBar
: Specifies the widget ID for the menu bar to be associated with the main window widget. You can set this ID only after creating an instance of the main window widget.

DwtNhorizontalScrollBar
: Specifies the scroll bar widget ID for the horizontal scroll bar in the main window widget. You can set

this ID only after creating an instance of the main window widget.

`DwtNverticalScrollBar`
Specifies the scroll bar widget ID for the vertical scroll bar in the main window widget. You can set this ID only after creating an instance of the main window widget.

`DwtNacceptFocus` Specifies a boolean value that, when `False`, indicates that the main window widget does not accept the input focus. When the main window widget is asked to accept the input focus, it attempts to give the input focus first to `DwtNworkWindow` and then to `DwtNcommandWindow`. If neither accepts the input focus and `DwtNacceptFocus` is `True`, the main window widget accepts the input focus.

`DwtNfocusCallback`
Specifies the callback function or functions called when the main window has accepted the input focus. For this callback, the reason is `DwtCRFocus`.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRFocus` The main window widget has received the input focus.

`DwtCRHelpRequested` The user selected help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on

`XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtMenu (3Dwt)

Name

DwtMenu, DwtMenuCreate, DwtMenuPulldownCreate, DwtMenuPopupCreate – Creates a menu widget to contain other menu items (subwidgets) for the display of application menus.

Creates a pull-down (pop-up) menu.

Creates a pop-up menu (MB2 only).

Syntax

Widget DwtMenu(*parent_widget*, *name*, *x*, *y*, *format*,
orientation, *entry_callback*, *map_callback*,
help_callback)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
int *format*;
unsigned char *orientation*;
DwtCallbackPtr *entry_callback*;
DwtCallbackPtr *map_callback*;
DwtCallbackPtr *help_callback*;

Widget DwtMenuCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Widget DwtMenuPulldownCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Widget DwtMenuPopupCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.

format Specifies the type of menu widget. You can pass `DwtMenuPopup`, `DwtMenuPulldown`, or `DwtMenuWorkArea`.

orientation Specifies whether the menu list is vertical or horizontal. You can pass `DwtOrientationHorizontal` or `DwtOrientationVertical`. This argument sets the `DwtNorientation` attribute associated with `DwtMenuCreate`.

entry_callback If this callback is defined, all menu entry activation callbacks are revectored to call back through this callback. If this callback is NULL, the individual menu entry callbacks work as usual. For this callback, the reason is `DwtCRActivate`. This argument sets the `DwtNentryCallback` attribute associated with `DwtMenuCreate`.

map_callback Specifies the callback function or functions called when the window is about to be mapped. For this callback, the reason is `DwtCRMap`. The *map_callback* argument is supported only if *format* is `DwtMenuPopup` or `DwtMenuPulldown`. The *map_callback* argument is ignored if *format* is `DwtMenuWorkArea`.

This argument sets the `DwtNmapCallback` attribute associated with `DwtMenuCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtMenu` and `DwtMenuCreate` functions create an instance of a menu widget and return its associated widget ID. The `DwtMenuPulldownCreate` function creates an instance of a pull-down menu widget and returns its associated widget ID. The `DwtMenuPopupCreate` function creates an instance of a pop-up menu widget and returns its associated widget ID. A menu is a composite widget that contains other widgets (push buttons, pull-down menus, toggle buttons, labels, and separators). The subwidgets handle most I/O that display information and query the user for input. The menu widget provides no input semantics over and above the semantics of its subwidgets. The menu widget works with these widget subclasses: push buttons, toggle buttons, pull-down menu entries, labels, and separators. If `DwtNentryCallback` is non-NULL when activated, all subwidgets call back to this callback. Otherwise, the individual subwidgets handle the activated callbacks.

Inherited Attributes

The following table lists the attributes inherited by the menu widget.

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	If menu orientation is `DwtOrientationVertical`, default is the maximum entry `DwtNwidth` or 16 pixels. If menu orientation is `DwtOrientationHorizontal`, default is the sum of `DwtNwidth` and `DwtNspacing` or 16 pixels.

DwtNheight	Dimension	If menu orientation is DwtOrientationVertical, default is the sum of DwtNheight and DwtNspacing or 16 pixels. If menu orientation is DwtOrientationHorizontal, default is the maximum entry DwtNheight or 16 pixels.
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True Setting the sensitivity of the menu causes all widgets contained in that menu to be set to the same sensitivity.
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

The following table lists the attributes inherited by the pull-down menu and pop-up menu widgets.

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	For DwtMenuPopupCreate, determined by the geometry manager For DwtMenuPulldownCreate, this attribute is not supported
DwtNy	Position	For DwtMenuPopupCreate, determined by the geometry manager For DwtMenuPulldownCreate, this attribute is not supported
DwtNwidth	Dimension	Set as large as necessary to hold all child widgets
DwtNheight	Dimension	Set as large as necessary to hold all child widgets
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True

DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Menu Attributes

DwtNspacing	Dimension	Zero pixels
DwtNmarginHeight	Dimension	3 pixels
DwtNmarginWidth	Dimension	Three pixels
DwtNorientation	unsigned char	DwtOrientationVertical
DwtNadjustMargin	Boolean	True
DwtNentryBorder	short	Zero pixels
DwtNmenuAlignment	Boolean	True
DwtNentryAlignment	unsigned char	DwtAlignmentBeginning
DwtNmenuPacking	unsigned char	DwtMenuPackingTight (for all menu types except for radio boxes) DwtMenuPackingColumn (for radio boxes)
DwtNmenuNumColumns	short	One row or column
DwtNmenuRadio	Boolean	False True (for radio boxes)
DwtNradioAlwaysOne	Boolean	True
DwtNmenuIsHomogeneous	Boolean	False

		True (for radio boxes)
DwtNmenuEntryClass	WidgetClass	NULL Radio boxes, however, default to the togglebuttonwidgetclass.
DwtNmenuHistory	Widget	Zero
DwtNentryCallback	DwtCallbackPtr	NULL
DwtNmenuHelpWidget	Widget	NULL
DwtNchangeVisAtts	Boolean	True
DwtNmenuExtendLastRow	Boolean	True

Widget-Specific Attributes

The following table lists the widget-specific attributes for the menu widget. Descriptions of these attributes follow the table.

Attribute Name	Data Type	Default
DwtNspacing	Dimension	Zero pixels
DwtNmarginHeight	Dimension	3 pixels
DwtNmarginWidth	Dimension	Three pixels
DwtNorientation	unsigned char	DwtOrientationVertical
DwtNadjustMargin	Boolean	True
DwtNentryBorder	short	Zero pixels
DwtNmenuAlignment	Boolean	True
DwtNentryAlignment	unsigned char	DwtAlignmentBeginning
DwtNmenuPacking	unsigned char	DwtMenuPackingTight (for all menu types except for radio boxes) DwtMenuPackingColumn (for radio boxes)
DwtNmenuNumColumns	short	One row or column
DwtNmenuRadio	Boolean	False True (for radio boxes)
DwtNradioAlwaysOne	Boolean	True
DwtNmenuIsHomogeneous	Boolean	False True (for radio boxes)
DwtNmenuEntryClass	WidgetClass	NULL Radio boxes, however, default to the togglebuttonwidgetclass.
DwtNmenuHistory	Widget	Zero
DwtNentryCallback	DwtCallbackPtr	NULL
DwtNmenuHelpWidget	Widget	NULL
DwtNchangeVisAtts	Boolean	True

DwtNmenuExtendLastRow	Boolean	True

DwtNspacing
: Specifies in pixels the spacing between menu bar entry windows.

DwtNmarginHeight
: Specifies the number of pixels remaining around the entries. The height is the number of blank pixels above the first entry and below the last entry (for vertical menus).

DwtNmarginWidth
: Specifies the number of pixels remaining around the entries. The width is the number of blank pixels between the left and right edges of the menu and the border of the entries.

DwtNorientation
: Specifies whether the menu list is vertical or horizontal. You can pass DwtOrientationHorizontal or DwtOrientationVertical.

DwtNadjustMargin
: Specifies a boolean value that indicates whether the inner minor dimension margins of all entries should be set to the same value.

  All label subclass widgets have two types of margins. The two outer margins (DwtNmarginWidth and DwtNmarginHeight) are symmetrical about the center of the widget. The number of pixels specified in DwtNmarginWidth are blank to the right and the left of the widget. The four inner margins (DwtNmarginLeft, DwtNmarginRight, DwtNmarginTop, and DwtNmarginBottom) specify the number of pixels to leave on each side inside the outer margins.

  The outer margins are used to accommodate such things as the border highlighting of widgets. The inner margins are used to accommodate such things as pull-down widget hot spots and toggle button indicators.

  If True, all entries in a given column or row will have exactly the same minor dimension margins. (If DwtNorientation is

DwtOrientationHorizontal, the minor dimension is vertical; if DwtNorientation is DwtOrientationVertical, the minor dimension is horizontal.) All margins will have the value of the largest individual margin in the group. This keeps the left edge of text lined up, regardless of whether some entries have toggle indicators.

DwtNentryBorder Specifies the border width of windows on the entry widgets.

DwtNmenuAlignment
Specifies a boolean value that, when True, indicates all entries are aligned. If False, entry alignment is unchanged. This is applied only to subclasses of labelwidgetclass.

DwtNentryAlignment
Specifies the type of label alignment that is enforced for all entries when DwtNmenuAlignment is True. You can pass DwtAlignmentCenter (center alignment), DwtAlignmentBeginning (alignment at the beginning), or DwtAlignmentEnd (alignment at the end).

DwtNmenuPacking Specifies how to pack the entries of a menu into the whole menu. The value of DwtNorientation determines the major dimension. You can pass DwtMenuPackingTight, DwtMenuPackingColumn, or DwtNmenuPackingNone.

DwtMenuPackingTight indicates that given the current major dimension of the menu, entries are placed one after the other until the menu must wrap. When the menu wraps, it extends in the minor dimension as many times as required.

Each entry's major dimension is left unaltered; its minor dimension is set to the same value as the greatest entry in that particular row or column. Note that the minor dimension of any particular row or column is independent of other rows or columns.

DwtMenuPackingColumn indicates that all entries are placed in identically sized boxes. The box

is based on the size of the largest entry while the value of DwtNmenuNumColumns determines how many boxes are placed in the major dimension before extending in the minor dimension.

DwtNmenuPackingNone indicates that no packing is performed. The DwtNx and DwtNy attributes of each entry are left alone and the menu attempts to become large enough to enclose all entries.

DwtNmenuNumColumns

Specifies the number of minor dimension extensions that will be made to accommodate the entries. This attribute is used only if DwtNmenuPacking is set to DwtMenuPackingColumn.

For menus with an orientation of DwtOrientationVertical, this attribute indicates how many columns will be built. The number of entries per column will be adjusted to maintain this number of columns (if possible). For menus with an orientation of DwtOrientationHorizontal, this attribute indicates how many rows will be built.

DwtNmenuRadio

Specifies a boolean value that, when True, indicates that when one button is already on and another button is turned on, the first button is turned off automatically.

DwtNradioAlwaysOne

Specifies a boolean value that indicates if the radio button exclusivity should also ensure that one button must always be on. If True, when the only radio button on is turned off, it will automatically be turned back on. Note that this attribute has no effect unless DwtNmenuRadio is True.

DwtNmenuIsHomogeneous

Specifies a boolean value that indicates if the menu should enforce exact homogeneity among the children of this menu. If True, only the DwtNmenuEntryClass class (not subclass but exact class) will be allowed as children of this menu.

DwtNmenuEntryClass

Specifies the only widget class that can be added to the menu. For this to occur, the DwtNmenuIsHomogeneous attribute must be True. All other widget classes will not be added to the menu.

DwtNmenuHistory
Holds the widget ID of the last menu entry that was activated. If DwtNmenuRadio is True, DwtNmenuHistory holds the widget ID of the last toggle button to change from off to on. This attribute may be set to precondition option menus and pop-up menus

DwtNentryCallback
If this callback is defined, all menu entry activation callbacks are revectored to call back through this callback. If this callback is NULL, the individual menu entry callbacks work as usual. For this callback, the reason is DwtCRActivate.

DwtNmenuHelpWidget
If non-NULL, the help menu widget points to the menu item to be placed in the lower right corner of the menu bar.

DwtNchangeVisAtts
Specifies a boolean value that, when True, indicates that a menu widget can optionally make these changes to its children: (1) Set the border to a uniform widget; (2) align labels; (3) make margins for the border highlight at least 2 pixels wide; (4) set the indicator shape to oval for toggle buttons in radio boxes; (5) set DwtNvisibleWhenOff to False for toggle buttons.

When DwtNchangeVisAtts is False, a menu widget cannot make any of these changes.

DwtNmenuExtendLastRow
Specifies the boolean value that indicates whether the active area of each menu entry extends to the width of the menu (for vertical menus) or the height of the menu (for horizontal menus).

If True for vertical menus, all menu entries extend to the menu width; if False, menu entries vary in

length depending on the length of the label in the menu entry. If `True` for horizontal menus, all menu entries extend to the menu height; if `False`, menu entries vary in height, depending on the length of the label in the menu entry.

The following table lists the widget-specific attributes for the pull-down and pop-up menu widgets. Descriptions of these attributes follow the table.

Attribute Name	Data Type	Default
`DwtNmapCallback`	`DwtCallbackPtr`	NULL
`DwtNunmapCallback`	`DwtCallbackPtr`	NULL

`DwtNmapCallback` Specifies the callback function or functions called when the menu is mapped.

`DwtNunmapCallback`
Specifies the callback function or functions called when the menu is unmapped.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                int reason;
                XEvent *event;
                Widget s_widget;
                char *s_tag;
                char *s_callbackstruct;
} DwtMenuCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRActivate` The user selected a menu entry.

`DwtCRMap` The menu window is about to be mapped.

`DwtCRUnmap` The menu window was just unmapped.

`DwtCRHelpRequested` The user selected help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*. The s_widget member is set to the ID of the activating subwidget. The s_tag member is set to the tag supplied by the application programmer when the subwidget callback function was specified. The s_callbackstruct member is set to the subwidget's callback structure.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtMenuBar, DwtMenuBarCreate – Creates a menu bar widget to contain menus.

Syntax

Widget DwtMenuBar(*parent_widget*, *name*, *entry_callback*, *help_callback*)
Widget *parent_widget*;
char **name*;
DwtCallbackPtr *entry_callback*;
DwtCallbackPtr *help_callback*;

Widget DwtMenuBarCreate (*parent_widget*, *name*, *override_arglist*, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

entry_callback If this callback is defined, all menu entry activation callbacks are revectored to call back through this callback. If this callback is NULL, the individual menu entry callbacks work as usual. For this callback, the reason is `DwtCRActivate`. This argument sets the `DwtNentryCallback` attribute associated with `DwtMenuCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtMenuBar` and `DwtMenuBarCreate` functions create an instance of the menu bar widget and return its associated widget ID. When calling `DwtMenuBar`, you set the menu bar widget attributes presented in the formal parameter list. For `DwtMenuBarCreate`, you specify a list of attribute name/value pairs that represent all the possible menu bar widget attributes.

A menu bar widget is a composite widget that contains pull-down menu entry subwidgets. The subwidgets handle most of the I/O activity that display information and query the user for input. The menu bar widget provides no input semantics over and above those provided by its subwidgets.

If the menu bar does not have enough room to fit all its subwidgets on a single line, the menu bar attempts to wrap the remaining entries onto additional lines (if allowed by the geometry manager of the parent widget).

The menu bar widget works with these widget classes: pull-down menu entries, labels, and separators.

If `DwtNentryCallback` is not NULL when it is activated, all subwidgets call back to this callback. Otherwise, the individual subwidgets handle the activation callbacks.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	16 pixels
`DwtNheight`	`Dimension`	Number of lines needed to display all entries
`DwtNborderWidth`	`Dimension`	One pixel

DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True Note that setting the sensitivity of the menu bar causes all widgets contained in that menu bar to be set to the same sensitivity as the menu bar.
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	Used only by gadget children
DwtNhelpCallback	DwtCallbackPtr	NULL

Menu Attributes

DwtNspacing	Dimension	One pixel
DwtNmarginHeight	Dimension	Zero pixels
DwtNmarginWidth	Dimension	Three pixels
DwtNorientation	unsigned char	DwtOrientationVertical
DwtNadjustMargin	Boolean	True
DwtNentryBorder	short	Zero pixels

DwtNmenuAlignment	Boolean	True
DwtNentryAlignment	unsigned char	DwtAlignmentBeginning
DwtNmenuPacking	unsigned char	DwtMenuPackingTight (for all menu types except for radio boxes) DwtMenuPackingColumn (for radio boxes)
DwtNmenuNumColumns	short	One row or column
DwtNmenuRadio	Boolean	False True (for radio boxes)
DwtNradioAlwaysOne	Boolean	True
DwtNmenuIsHomogeneous	Boolean	False True (for radio boxes)
DwtNmenuEntryClass	WidgetClass	NULL Radio boxes, however, default to the togglebuttonwidgetclass.
DwtNmenuHistory	Widget	Zero
DwtNentryCallback	DwtCallbackPtr	NULL
DwtNmenuHelpWidget	Widget	NULL
DwtNchangeVisAtts	Boolean	True
DwtNmenuExtendLastRow	Boolean	True

Widget-Specific Attributes

The menu bar widget does not currently support any widget-specific attributes.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
                    Widget s_widget;
                    char *s_tag;
                    char *s_callbackstruct;
} DwtMenuCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRActivate	The user selected a menu entry.
DwtCRMap	The menu window is about to be mapped.
DwtCRUnmap	The menu window was just unmapped.
DwtCRHelpRequested	The user selected help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*. The s_widget member is set to the ID of the activating subwidget. The s_tag member is set to the tag supplied by the application programmer when the subwidget callback function was specified. The s_callbackstruct member is set to the subwidget's callback structure.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtMenuPosition (3Dwt)

Name

DwtMenuPosition – Positions menu when user presses MB2.

Syntax

void DwtMenuPosition(*position*, *event*)
Widget *position*;
XEvent **event*;

Arguments

position	Specifies the position of the menu.
event	Specifies the event passed to the action procedure which manages the pop-up menu.

Description

The `DwtMenuPosition` function positions the menu when the user presses MB2. This must be called before managing the pop-up menu.

See Also

DwtPullDownMenuEntryHilite (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtMessageBox, DwtMessageBoxCreate – Creates a message box widget for the application to display text to the user.

Syntax

Widget DwtMessageBox(*parent_widget*, *name*, *default_position*,
x, *y*, *style*, *ok_label*, *label*,
callback, *help_callback*)
Widget *parent_widget*;
char **name*;
Boolean *default_position*;
Position *x*, *y*;
int *style*;
DwtCompString *ok_label*, *label*;
DwtCallbackPtr *callback*;
DwtCallbackPtr *help_callback*;

Widget DwtMessageBoxCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, causes `DwtNx` and `DwtNy` to be ignored and forces the default widget position. The default widget position is centered in the parent window. If `False`, the specified `DwtNx` and `DwtNy` attributes are used to position the widget. This argument sets the `DwtNdefaultPosition` attribute associated with `DwtDialogBoxCreate`.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget

attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.

style Specifies the style of the dialog box widget. You can pass `DwtModal` (modal) or `DwtModeless` (modeless). This argument sets the `DwtNstyle` attribute associated with `DwtDialogBoxPopupCreate`.

label Specifies the text in the message line or lines. This argument sets the `DwtNlabel` attribute associated with `DwtMessageBoxCreate`.

ok_label Specifies the label for the Ok push button. If the label is a NULL string, the button is not displayed. This argument sets the `DwtNokLabel` attribute associated with `DwtMessageBoxCreate`.

callback Specifies the callback function or functions called when the user activates the OK push button. This argument sets the `DwtNyesCallback` attribute associated with `DwtMessageBoxCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtMessageBox` and `DwtMessageBoxCreate` functions create an instance of the message box widget and return its associated widget ID. When calling `DwtMessageBox`, you set the message box attributes presented in the formal parameter list. For `DwtMessageBoxCreate`, however, you specify a list of attribute name/value pairs that represent all the possible message box widget attributes.

The DwtMessageBoxCreate function conforms to the *XUI Style Guide* by providing optional secondary text below the primary text. This function also supports alignment mode for both the DwtNlabelAlignment and DwtNsecondLabelAlignment attributes.

The message box widget is a dialog box that allows the application to display informational messages to the user. You call this function to create a message box when the user does something unexpected, or when your application needs to display information to the user. The message box widget may contain an OK push button. When the style is DwtModal, the message box freezes the application and requires the user to explicitly dismiss the message box before the application proceeds. If the style is DwtModal when the user selects the OK push button, the widget is cleared from the screen but not destroyed. You can redisplay the widget by calling XtManageChild.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	5 pixels
DwtNheight	Dimension	5 pixels
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True

DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Dialog Pop-Up Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	NOT SUPPORTED	
DwtNunits	NOT SUPPORTED	
DwtNtitle	DwtCompString	Widget name
DwtNstyle	unsigned char	DwtModal
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	NOT SUPPORTED	
DwtNmarginWidth	Dimension	12 pixels
DwtNmarginHeight	Dimension	10 pixels
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	NOT SUPPORTED	
DwtNresize	unsigned char	DwtResizeShrinkWrap
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	NOT SUPPORTED	
DwtNcancelButton	NOT SUPPORTED	

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNokLabel	DwtCompString	"Acknowledged"
DwtNyesCallback	DwtCallbackPtr	NULL
DwtNsecondLabel	DwtCompString	NULL
DwtNlabelAlignment	unsigned char	DwtAlignmentCenter

DwtNsecondLabelAlignment	unsigned char	DwtAlignmentBeginning
DwtNiconPixmap	Pixmap	The default is the standard icon provided for each message-class widget as follows: (1) the default caution box icon is an exclamation point; (2) the default message box icon is an asterisk; (3) the default work box icon is the wait cursor (watch). See the *XUI Style Guide* for illustrations of the icons for each message class widget.

DwtNlabel Specifies the text in the message line or lines.

DwtNokLabel Specifies the label for the Ok push button. If the label is a NULL string, the button is not displayed.

DwtNyesCallback Specifies the callback function or functions called when the user clicks on the Yes button. For this callback, the reason is DwtCRYes.

DwtNsecondLabel Specifies the text for the secondary label. If the application specifies a second label and then wants to remove it, it should use XtSetValues to set DwtNsecondLabel to NULL or to an empty compound-string.

DwtNlabelAlignment
Specifies the alignment for the primary label. You can pass DwtAlignmentCenter (center alignment), DwtAlignmentBeginning (alignment at the beginning), or DwtAlignmentEnd (alignment at the end).

DwtNsecondLabelAlignment
Specifies the alignment for the secondary label. You can pass DwtAlignmentCenter (center alignment), DwtAlignmentBeginning (alignment at the beginning), or DwtAlignmentEnd (alignment at the end).

DwtNiconPixmap Specifies the pixmap used for the icon.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRYes`	The user activated the Yes button.
`DwtCRFocus`	The message box has received the input focus.
`DwtCRHelpRequested`	The user selected Help somewhere in the message box.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtOpenHierarchy – Allocates a hierarchy ID and opens all the UID files in the hierarchy.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtOpenHierarchy(num_files, file_names_list,
ancillary_structures_list,
                    hierarchy_id_return)
    DRMCount num_files;
    String file_names_list [];
    IDBOSOpenParamPtr *ancillary_structures_list;
    DRMHierarchy *hierarchy_id_return;
```

Arguments

num_files Specifies the number of files in the name list.

file_names_list Specifies an array of pointers to character strings that identify the `.uid` files.

ancillary_structures_list
A list of operating system-dependent ancillary structures corresponding to such things as file names, clobber flag, and so forth. This argument should be NULL for most operations. If you need to reference this structure, see the definition of `IDBOSOpenParamPtr` in `DwtAppl.h` for more information.

hierarchy_id_return
Returns the search hierarchy ID. The search hierarchy ID identifies the list of `.uid` files that DRM will search (in order) when performing subsequent fetch calls.

Description

The `DwtOpenHierarchy` function allows the user to specify the list of UID files that DRM will search in subsequent fetch operations. All subsequent fetch operations will return the first occurrence of the named item encountered while traversing the UID hierarchy from the first list element (UID file specification) to the last list element. This function also allocates a hierarchy ID and opens all the UID files in the hierarchy. It initializes the optimized search lists in the hierarchy. If `DwtOpenHierarchy`

encounters any errors during its execution, any files that were opened are closed.

Each UID file specified in *file_names_list* can specify either a full directory pathname or a file name. If a UID file does not specify the pathname it will not contain any embedded slashes (/), and it will be accessed through the UIDPATH environment variable.

The UIDPATH environment variable specifies search paths and naming conventions associated with UID files. It can contain the substitution fields %L and %N, where the current setting of the LANG environment variable is substituted for %L and the .uid name passed to `DwtOpenHierarchy` is substituted for %N. For example, the following UID path and `DwtOpenHierarchy` call would cause DRM to open two separate `.uid` files:

```
UIDPATH=/uidlib/%L/%N.uid:/uidlib/%N/%L
 static char *uid_files[] = {"/usr/users/me/test.uid", "test2"};
  DRMHierarchy  *Hierarchy_id;
  DwtOpenHierarchy((DRMCount)2,uid_files, NULL, Hierarchy_id)
```

The first file, `/usr/users/me/test.uid`, would be opened as specified, as this file specification includes a pathname. The second file, `test2`, would be looked for first in `/uidlib/$LANG/test2.uid`, and second in `/uidlib/test2/$LANG`.

After `DwtOpenHierarchy` opens the UID hierarchy, you should not delete or modify the UID files until you close the UID hierarchy by calling `DwtCloseHierarchy`.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMNotFound`	File not found.
`DRMFailure`	The function failed.

See Also

DwtCloseHierarchy(3Dwt)

Name

DwtOptionMenu, DwtOptionMenuCreate – Creates an option menu widget to display and handle an application option list of attributes or modes of the menu topic. It allows just one option selected from the list in the menu.

Syntax

Widget DwtOptionMenu(*parent_widget*, *name*, *x*, *y*,
label, *sub_menu_id*,
entry_callback, *help_callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString *label*;
Widget *sub_menu_id*;
DwtCallbackPtr *entry_callback*, *help_callback*;

Widget DwtOptionMenuCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
label	Specifies the text in the menu label. This argument sets the `DwtNlabel` attribute associated with `DwtMenuCreate`.
sub_menu_id	Specifies the widget ID of the pull-down menu associated with the option menu during the creation phase.

entry_callback If this callback is defined, all menu entry activation callbacks are revectored to call back through this callback. If this callback is NULL, the individual menu entry callbacks work as usual. For this callback, the reason is `DwtCRActivate`. This argument sets the `DwtNentryCallback` attribute associated with `DwtMenuCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtOptionMenu` and `DwtOptionMenuCreate` functions create an instance of the option menu widget and return its associated widget ID. When calling `DwtOptionMenu`, you set the option menu widget attributes presented in the formal parameter list. For `DwtOptionMenuCreate`, however, you specify a list of attribute name/value pairs that represent all the possible option menu widget attributes. The option menu widget is a composite widget containing other subwidgets (toggle button widgets). It displays and handles an application option list of attributes or modes of the menu topic. Basically, the option menu consists of a label identifying the menu and an active area to the right. This composite widget contains other subwidgets (toggle button widgets) in the active area. It displays the current option selected, and, on request, generates a pop-up menu with specific options available. In addition, it ensures that a user can select only one choice at any given time.

If `DwtNentryCallback` is non-NULL, then all the toggle button callbacks will execute the *entry_callback* function, rather than the procedure specified in the toggle. Otherwise, if `DwtNentryCallback` is NULL, then the individual callbacks work as usual.

Option menus also position the pop-up part of the menu so that the menu history widget covers the selection part of the option menu. Option menus also copy the label of the menu history widget into the selection part.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Set as large as necessary to hold all child widgets
DwtNheight	Dimension	Set as large as necessary to hold all child widgets
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL
Common Attributes		
DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown

DwtNfont	DwtFontList	The default XUI Toolkit font Used only by gadget children
DwtNhelpCallback	DwtCallbackPtr	NULL
Menu Attributes		
DwtNspacing	Dimension	Zero pixels
DwtNmarginHeight	Dimension	3 pixels
DwtNmarginWidth	Dimension	Three pixels
DwtNorientation	unsigned char	DwtOrientationVertical
DwtNadjustMargin	Boolean	True
DwtNentryBorder	short	Zero pixels
DwtNmenuAlignment	Boolean	True
DwtNentryAlignment	unsigned char	DwtAlignmentBeginning
DwtNmenuPacking	unsigned char	DwtMenuPackingTight (for all menu types except for radio boxes) DwtMenuPackingColumn (for radio boxes)
DwtNmenuNumColumns	short	One row or column
DwtNmenuRadio	Boolean	False True (for radio boxes)
DwtNradioAlwaysOne	Boolean	True
DwtNmenuIsHomogeneous	Boolean	False True (for radio boxes)
DwtNmenuEntryClass	WidgetClass	NULL Radio boxes, however, default to the togglebuttonwidgetclass.
DwtNmenuHistory	Widget	Zero
DwtNentryCallback	DwtCallbackPtr	NULL
DwtNmenuHelpWidget	Widget	NULL
DwtNchangeVisAtts	Boolean	True
DwtNmenuExtendLastRow	Boolean	True

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNsubMenuId	Widget	Zero

DwtNlabel — Specifies the label that will be placed to the left of the current value.

DwtNsubMenuId	Specifies the widget ID of the pull-down menu associated with the option menu during the creation phase.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
            int reason;
            XEvent *event;
            Widget s_widget;
            char *s_tag;
            char *s_callbackstruct;
} DwtMenuCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRActivate	The user selected a menu entry.
DwtCRHelpRequested	The user selected help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*. The s_widget member is set to the ID of the activating subwidget. The s_tag member is set to the tag supplied by the application programmer when the subwidget callback function was specified. The s_callbackstruct member is set to the subwidget's callback structure.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtPullDownMenuEntry (3Dwt)

Name

DwtPullDownMenuEntry, DwtPullDownMenuEntryCreate – Creates an instance of the pull-down menu entry widget.

Syntax

Widget DwtPullDownMenuEntry(*parent_widget*, *name*,
x, *y*, *label*,
menu_id, *callback*, *help_callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString *label*;
Widget *menu_id*;
DwtCallbackPtr *callback*, *help_callback*;

Widget DwtPullDownMenuEntryCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
label	Specifies the text of the label entry in the parent menu. This argument sets the `DwtNlabel` attribute associated with `DwtLabelCreate`.
menu_id	Specifies the ID of the pull-down menu widget.

callback Specifies the callback function or functions called back when a button inside a pull-down menu entry widget is activated. This argument sets the `DwtNactivateCallback` and `DwtNpullingCallback` attributes associated with `DwtPullDownMenuEntryCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtPullDownMenuEntry` and `DwtPullDownMenuEntryCreate` functions create an instance of the pull-down menu entry widget and return its associated widget ID. When calling `DwtPullDownMenuEntry`, you set the pull-down menu entry widget attributes presented in the formal parameter list. For `DwtPullDownMenuEntryCreate`, however, you specify a list of attribute name/value pairs that represent all the possible pull-down menu entry widget attributes.

A pull-down menu entry widget is made up of two parts: a label (within the parent menu) and a select area or "hotspot." The hotspot is the full widget window. Otherwise, the hotspot is a separate rectangle on the right side of the entry label.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager

DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	The DwtNlabel width, plus the DwtNhotSpotPixmap width or the DwtNpixmap width, plus DwtNmarginWidth times two
DwtNheight	Dimension	The DwtNlabel or DwtNpixmap height, plus DwtNmarginHeight times two
DwtNborderWidth	Dimension	zero pixels
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Label Attributes

DwtNlabelType	unsigned char	DwtCString
DwtNlabel	DwtCompString	Widget name
DwtNmarginWidth	Dimension	Two pixels for text Zero pixels for pixmap
DwtNmarginHeight	Dimension	Two pixels for text Zero pixels for pixmap
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNpixmap	Pixmap	NULL
DwtNmarginLeft	Dimension	Zero
DwtNmarginRight	Dimension	Zero
DwtNmarginTop	Dimension	Zero
DwtNmarginBottom	Dimension	Zero
DwtNconformToText	Boolean	True, if the widget is created with a width and height of zero False, if the widget is created with a non-zero width and height

Widget-Specific Attributes

You can set the following widget-specifc attributes in the *override_arglist*:

Attribute Name	Data Type	Default
DwtNsubMenuId	Widget	NULL
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNpullingCallback	DwtCallbackPtr	NULL
DwtNhotSpotPixmap	Pixmap	NULL

DwtNsubMenuId
: Specifies the widget ID of the submenu that will be displayed when the pull-down menu is activated.

DwtNactivateCallback
: Specifies the callback that is executed when the user releases a button inside the pull-down menu widget. For this callback, the reason is DwtCRActivate.

DwtNpullingCallback
: Specifies the callback function or functions called just prior to pulling down the submenu. This callback occurs just before the submenu's map callback. You can use this callback to defer the

creation of the submenu. For this callback, the reason is DwtCRActivate.

DwtNhotSpotPixmap
Specifies the pixmap to use for the hotspot icon.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRActivate	The user selected the pull-down menu entry.
DwtCRHelpRequested	The user selected Help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtPullDownMenuEntryHilite – Highlights a menu entry.

Syntax

void DwtPullDownMenuEntryHilite(*pulldown*, *highlight*)
Widget *pulldown*;
int *highlight*;

Arguments

position	Specifies the pulldown menu..
highlight	Specifies whether a menu entry is highlighted. If the value is one, the entry is highlighted. If the value is zero, the entry is not highlighted.

Description

The `DwtPullDownMenuEntryHilite` function keeps an entry highlight after the user clicks on a menu item.

See Also

DwtMenuPosition (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtPullEntryGadgetCreate (3Dwt)

Name

DwtPullEntryGadgetCreate – Creates a pull-down menu entry gadget.

Syntax

```
Widget DwtPullEntryGadgetCreate (parent_widget, name,
                                 override_arglist,
                                 override_argcount)
    Widget parent_widget;
    char *name;
    ArgList override_arglist;
    int override_argcount;
```

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtPullEntryGadgetCreate` function creates an instance of the pull-down menu entry gadget and returns its associated gadget ID.

A pull-down menu entry gadget is similar in appearance and semantics to a pull-down menu entry widget. Like all gadgets, it does not have a window but uses the window of the closest antecedent widget. This gadget must be a child of a menu class widget.

Because a pull-down menu entry gadget is not a subclass of composite, children are not supported.

The sizing of the gadget is affected by the font and the label.

Inherited Attributes

Attribute Name	Data Type	Default
Rectangle Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	The label width, plus the hotspot width, plus 2 times DwtNmarginWidth
DwtNheight	Dimension	The text label or pixmap label height plus 2 times DwtNmarginHeight
DwtNborderWidth	Dimension	Zero pixels
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
Label Gadget Attributes		
DwtNlabel	DwtCompString	Widget name
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNdirectionRToL	Boolean	False
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNsubMenuId	Widget	NULL
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNpullingCallback	DwtCallbackPtr	NULL

DwtNsubMenuId Specifies the widget ID of the submenu that will be displayed when the pull-down menu is activated.

DwtNactivateCallback
Specifies the callback that is executed when the user releases a button inside the pull-down menu widget.

For this callback, the reason is DwtCRActivate.

DwtNpullingCallback
Specifies the callback function or functions called just prior to pulling down the submenu. This callback occurs just before the submenu's map callback. You can use this callback to defer the creation of the submenu. For this callback, the reason is DwtCRActivate.

Return Value

This function returns the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRActivate	The user selected the pull-down menu entry.
DwtCRHelpRequested	The user selected Help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

DwtPullDownMenuEntry (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtPushButton, DwtPushButtonCreate – Creates a push button widget.

Syntax

Widget DwtPushButton(*parent_widget*, *name*, *x*, *y*,
label, *callback*, *help_callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString *label*;
DwtCallbackPtr *callback*, *help_callback*;

Widget DwtPushButtonCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
label	Specifies the push button label. This argument sets the `DwtNlabel` attribute associated with `DwtLabelCreate`.
callback	Specifies the callback function or functions called back when a push button is activated. This argument sets the `DwtNactivateCallback`, `DwtNarmCallback`, and `DwtNdisarmCallback` attributes associated with `DwtPushButtonCreate`.
help_callback	Specifies the callback function or functions called when a

help request is made. This argument sets the DwtNhelpCallback common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtPushButton and DwtPushButtonCreate functions create an instance of the push button widget and return its associated widget ID. When calling DwtPushButton, you set the push button widget attributes presented in the formal parameter list. For DwtPushButtonCreate, however, you specify a list of attribute name/value pairs that represent all the possible push button widget attributes.

The push button is a primitive widget that displays a rectangular border around a label. The label defines the immediate action of the button (for example, Ok or Cancel in a dialog box).

The sizing is affected by spacing, font (affects indicator), and label. See the description for DwtLabel and DwtLabelCreate.

The push button widget follows the same rules for geometry management as its superclass the label widget, which you create by calling DwtLabel or DwtLabelCreate. Like the label widget, the push button widget does not support children; therefore, it always refuses geometry requests.

The push button widget follows the same rules for resizing as its superclass the label widget, which you create by calling DwtLabel or DwtLabelCreate. Like the label widget, the push button widget does nothing on a resize by its parents.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager

DwtNwidth	Dimension	The width of the label or pixmap plus DwtNmarginWidth times two
DwtNheight	Dimension	The height of the label or pixmap plus DwtNmarginHeight times two
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Label Attributes

DwtNlabelType	unsigned char	DwtCString
DwtNlabel	DwtCompString	Widget name
DwtNmarginWidth	Dimension	Two pixels for text Zero pixels for pixmap

DwtNmarginHeight	Dimension	Two pixels for text Zero pixels for pixmap
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNpixmap	Pixmap	NULL
DwtNmarginLeft	Dimension	Zero
DwtNmarginRight	Dimension	Zero
DwtNmarginTop	Dimension	Zero
DwtNmarginBottom	Dimension	Zero
DwtNconformToText	Boolean	True, if the widget is created with a width and height of zero False, if the widget is created with a non-zero width and height

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNbordHighlight	Boolean	False
DwtNfillHighlight	Boolean	False
DwtNshadow	Boolean	True
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNarmCallback	DwtCallbackPtr	NULL
DwtNdisarmCallback	DwtCallbackPtr	NULL
DwtNacceleratorText	DwtCompString	NULL
DwtNbuttonAccelerator	char *	NULL
DwtNinsensitivePixmap	Pixmap	NULL

DwtNbordHighlight
: Specifies a boolean value that, when True, highlights the border.

DwtNfillHighlight
: Specifies a boolean value that, when True, fills the highlighted button.

DwtNshadow
: Specifies whether the shadow of the push button is displayed.

DwtNactivateCallback
: Specifies the callback function or functions called when the push button is activated. The button is activated when the user presses and releases MB1 while the pointer is inside the push button widget.

Activating the push button also disarms the push button. For this callback, the reason is DwtCRActivate.

DwtNarmCallback Specifies the callback function or functions called when the push button is armed. The push button is armed when the user presses and releases MB1 while the pointer is inside the push button widget. For this callback, the reason is DwtCRArm.

DwtNdisarmCallback
Specifies the callback function or functions called when the push button is disarmed. The button is disarmed in two ways. After the user activates the button (presses and releases MB1 while the pointer is inside the push button widget), the button is disarmed. When the user presses MB1 while the pointer is inside the push button widget but moves the pointer outside the push button before releasing MB1, the button is disarmed. For this callback, the reason is DwtCRDisarm.

DwtNacceleratorText
Specifies the compound-string text displayed for the accelerator.

DwtNbuttonAccelerator
Sets an accelerator on a push button widget. This is the same as the DwtNtranslations core attribute except that only the left side of the table is to be passed as a character string, not compiled. The application is responsible for calling XtInstallAllAccelerators to install the accelerator where the application needs it.

DwtNinsensitivePixmap
Specifies the pixmap used when the push button is set to insensitive. This attribute applies only if the push button label is specified as a pixmap.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRActivate`	The user activated the push button by pressing MB1 while the pointer was inside the push button widget.
`DwtCRArm`	The user armed the push button by pressing MB1 while the pointer was inside the push button widget.
`DwtCRDisarm`	The user disarmed the push button in one of two ways. The user pressed MB1 while the pointer was inside the push button widget, but did not release it until after moving the pointer outside the push button widget. Or, the user activated the push button, which also disarms it.
`DwtCRHelpRequested`	The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtPushButtonGadgetCreate – Creates a push button gadget.

Syntax

Widget DwtPushButtonGadgetCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtLabelGadgetCreate` function creates an instance of the label gadget and returns its associated gadget ID. A label gadget is similar in appearance and semantics to a label widget. Like all gadgets, the label gadget does not have a window but uses the window of the closest antecedent widget. Thus, the antecedent widget provides all event dispatching for the gadget. This currently restricts gadgets to being descendents of menu or dialog class (or subclass) widgets. Drawing information such as font and color are also those of the closest antecedent widget.

Inherited Attributes

Attribute Name	Data Type	Default
Rectangle Attributes		

DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	The width of the label plus margins
DwtNheight	Dimension	The height of the label plus margins
DwtNborderWidth	Dimension	1 pixel
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	NULL
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNacceleratorText	DwtCompString	NULL
DwtNbuttonAccelerator	char *	NULL

DwtNlabel
: Specifies the push button label.

DwtNactivateCallback
: Specifies the callback function or functions called when the push button is activated. The button is activated when the user presses and releases MB1 while the pointer is inside the push button gadget. For this callback, the reason is DwtCRActivate.

DwtNacceleratorText
: Specifies the compound-string text displayed for the accelerator.

DwtNbuttonAccelerator
: Sets an accelerator on a push button widget. This is the same as the DwtNtranslations core attribute except that only the left side of the table is to be passed as a character string, not compiled. The application is responsible for calling XtInstallAllAccelerators to install the

accelerator where the application needs it.

Return Value

This function returns the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRActivate`	The user activated the push button.
`DwtCRHelpRequested`	The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtRadioBox (3Dwt)

Name

DwtRadioBox, DwtRadioBoxCreate – Creates a radio box widget for the application to display multiple toggle buttons.

Syntax

Widget DwtRadioBox(*parent_widget*, *name*, *x*, *y*,
entry_callback, *help_callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCallbackPtr *entry_callback*, *help_callback*;

Widget DwtRadioBoxCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
entry_callback	If this callback is defined, all menu entry activation callbacks are revectored to call back through this callback. If this callback is NULL, the individual menu entry callbacks work as usual. For this callback, the reason is `DwtCRActivate`. This argument sets the `DwtNentryCallback` attribute associated with `DwtMenuCreate`.
help_callback	Specifies the callback function or functions called when a

help request is made. This argument sets the DwtNhelpCallback common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The radio box is a composite widget that contains multiple toggle button widgets. The radio box arbitrates and ensures that only one toggle button is on at any one given time. When calling DwtRadioBox, you set the radio box widget attributes presented in the formal parameter list. For DwtRadioBoxCreate, however, you specify a list of attribute name/value pairs that represent all the possible radio box widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Set as large as necessary to hold all child widgets
DwtNheight	Dimension	Set as large as necessary to hold all child widgets
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True

		Setting the sensitivity of the radio box causes all widgets contained in that radio box to be set to the same sensitivity.
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Menu Attributes

DwtNspacing	Dimension	Zero pixels
DwtNmarginHeight	Dimension	3 pixels
DwtNmarginWidth	Dimension	Three pixels
DwtNorientation	unsigned char	DwtOrientationVertical
DwtNadjustMargin	Boolean	True
DwtNentryBorder	short	Zero pixels
DwtNmenuAlignment	Boolean	True
DwtNentryAlignment	unsigned char	DwtAlignmentBeginning
DwtNmenuPacking	unsigned char	DwtMenuPackingTight (for all menu types except for radio boxes) DwtMenuPackingColumn (for radio boxes)
DwtNmenuNumColumns	short	One row or column
DwtNmenuRadio	Boolean	False

		True (for radio boxes)
DwtNradioAlwaysOne	Boolean	True
DwtNmenuIsHomogeneous	Boolean	False True (for radio boxes)
DwtNmenuEntryClass	WidgetClass	NULL Radio boxes, however, default to the togglebuttonwidgetclass.
DwtNmenuHistory	Widget	Zero
DwtNentryCallback	DwtCallbackPtr	NULL
DwtNmenuHelpWidget	Widget	NULL
DwtNchangeVisAtts	Boolean	True
DwtNmenuExtendLastRow	Boolean	True

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
                    Widget s_widget;
                    char *s_tag;
                    char *s_callbackstruct;
} DwtRadioBoxCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRValueChanged	The user activated the toggle button to change state.
DwtCRMap	The radio box is about to be mapped.
DwtCRHelpRequested	The user selected Help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*. The s_widget member is set to the ID of the activating subwidget. The s_tag member is set to the tag supplied by the application programmer when the subwidget callback function was specified. The

s_callbackstruct member is set to the subwidget's callback structure.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtReCopyToClipboard – Copies a data item previously pased by name to the clipboard.

Syntax

```
int DwtReCopyToClipboard(display, window, data_id,
                         buffer, length, private_id)
    Display *display;
    Window window;
    int data_id;
    char *buffer;
    unsigned long length;
    int private_id;
```

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
data_id	Specifies an identifying number assigned to the data item that uniquely identifies the data item and the format. This number was assigned by `DwtCopyToClipboard` to the data item.
buffer	Specifies the buffer from which the clipboard copies the data.
length	Specifies the number of bytes in the data item.
private_id	Specifies the private data that the application wants to store with the data item.

Description

The `DwtReCopyToClipboard` function copies the actual data for a data item that was previously passed by name to the clipboard. Additional calls to `DwtReCopyToClipboard` append new data to the existing data. This function cannot be used to pass data by name.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtRegisterClass – Saves the information needed for DRM to access the widget creation function for user-defined widgets.

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtRegisterClass(class_code, class_name, create_name,
                          create_proc, class_record)
    DRMType class_code;
    String class_name;
    String create_name;
    Widget (* create_proc) ();
    WidgetClass class_record;
```

Arguments

class_code	Specifies the code name of the class. For all application-defined widgets, this code name is `DRMwcUnknown`. For all XUI Toolkit widgets, each code name begins with the letters `DRMwc`. The code names for all application widgets are defined in `DRM.h`.
class_name	Specifies the case-sensitive name of the class. The class names for all XUI Toolkit widgets are defined in `DRM.h`. Each class name begins with the letters `DRMwcn`.
create_name	Specifies the case-sensitive name of the low-level widget creation function for the class. An example from the XUI Toolkit is `DwtLabelCreate`. Arguments are *parent_widget*, *name*, *override_arglist*, and *override_argcount*. For user-defined widgets, *create_name* is the creation procedure in the UIL that defines this widget.
create_proc	Specifies the address of the creation function that you named in *create_name*.
class_record	Specifies a pointer to the class record.

DwtRegisterClass(3Dwt) ( )

Description

The `DwtRegisterClass` function allows DRM to access user-defined widget classes. This function registers the necessary information for DRM to create widgets of this class. You must call `DwtRegisterClass` prior to fetching any user-defined class widget.

`DwtRegisterClass` saves the information needed to access the widget creation function and to do type conversion of argument lists by using the information in DRM databases.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMFailure`	The allocation of the class descriptor failed.

Name

DwtRegisterDRMNames – Registers the values associated with the names referenced in UIL (for example, UIL callback function names or UIL identifier names).

Syntax

```
#include <X11/DwtAppl.h>
Cardinal DwtRegisterDRMNames(register_list, register_count)
    DRMRegisterArglist register_list;
    DRMCount register_count;
```

Arguments

register_list	Specifies a list of name/value pairs for the names to be registered. Each name is a case-sensitive, NULL-terminated ASCII string. Each value is a 32-bit quantity, interpreted as a procedure address if the name is a callback function, and uninterpreted otherwise.
register_count	Specifies the number of entries in *register_list*.

Description

The `DwtRegisterDRMNames` function registers a vector of names and associated values for access in DRM. The values can be callback functions, pointers to user-defined data, or any other values. The information provided is used to resolve symbolic references occurring in UID files to their run-time values. For callbacks, this information provides the procedure address required by the XUI Toolkit. For names used as identifiers in UIL, this information provides any run-time mapping the application needs.

The names in the list are case-sensitive. The list can be either ordered or unordered.

Callback functions registered through `DwtRegisterDRMNames` can be either regular or creation callbacks. Regular callbacks have declarations determined by XUI Toolkit and user requirements. Creation callbacks have the same format as any other callback:

```
void CallBackProc(widget_id, tag, callback_data)
        Widget *widget_id;
        Opaque tag;
        DwtAnyCallbackStruct *callback_data;
```

widget_id	Specifies the widget ID associated with the widget performing the callback (as in any callback function).
tag	Specifies the tag value (as in any callback function).
callback_data	Specifies a widget-specific data structure. This data structure has a minimum of two members: event and reason. The reason member is always set to `DwtCRCreate`.

Note that the widget name and parent are available from the widget record accessible through *widget_id*.

Return Value

This function returns one of these status return constants:

`DRMSuccess`	The function executed successfully.
`DRMFailure`	Memory allocation failed.

Name

DwtResolvePartOffsets – Allows writing of upward-compatible applications and widgets.

Syntax

void DwtResolvePartOffsets(*widget_class*, *offset*)
WidgetClass *widget_class*;
DwtOffsetPtr **offset*;

Arguments

widget_class	Specifies the widget class pointer for the created widget.
offset	Specifies the offset record.

Description

The use of offset records requires one extra global variable per widget class. The variable consists of a pointer to an array of offsets into the widget record for each part of the widget structure. The `DwtResolvePartOffsets` function allocates the offset records needed by an application to guarantee upward-compatible applications and widgets. These offset records are used by the widget to access all of the widget's variables. A widget needs to take the following steps:

- Instead of creating a resource list, the widget creates an offset resource list. To help you accomplish this, use the `DwtPartResource` structure and the `DwtPartOffset` macro. The `DwtPartResource` data structure looks just like a resource list, but instead of having one integer for its offset, it has two shorts. This gets put into the class record as if it were a normal resource list. Instead of using `XtOffset` for the offset, it uses `DwtPartOffset`.
- Instead of putting the widget size in the class record, the widget puts the widget part in the same field.
- Instead of putting `XtVersion` in the class record, the widget puts XtVersionDontCheck in the class record.
- The widget defines a variable to point to the offset record. This can be part of the widget's class record or a separate global variable.
- In class initialization, the widget calls `DwtResolvePartOffsets`, passing it the offset address and the class record. This does several things:

- Adds the superclass (which, by definition, has already been initialized) size field to the part size field.
- Allocates an array based upon the number of superclasses.
- Fills in the offsets of all the widget parts with the appropriate values, determined by examining the size fields of all superclass records.
- Uses the part offset array to modify the offset entries in the resource list to be real offsets, in place.
- Instead of accessing fields directly, the widget must always go through the offset table. You will probably define macros for each field to make this easier. Assume an integer field "xyz":

```
#define BarXyz(w) (*(int *)(((char *) w) + offset[BarIndex] +
            XtOffset(BarPart,xyz)))
```

The `DwtField` macro helps you access these fields. Because the `DwtPartOffset` and `DwtField` macros concatenate arguments, you must ensure there is no space before or after the part argument. For example, the following do not work because of the space before or after the part (Label) argument:

```
DwtField(w, offset, Label, text, char *)
DwtPartOffset( Label, text).
```

Therefore, you must not have any spaces before or after the part (Label) argument, as illustrated here:

```
DwtField(w, offset,Label, text, char *)
```

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtScale, DwtScaleCreate – Creates a scale widget that allows an application to display a scale for vernier control of a parameter while displaying the current value and range.

Syntax

```
Widget DwtScale(parent_widget, name, x, y,
                width, height, scale_width, scale_height,
                title, min_value, max_value, decimal_points,
                value, orientation, callback,
                drag_callback, help_callback)
    Widget parent_widget;
    char *name;
    Position x, y;
    Dimension width, height;
    Dimension scale_width, scale_height;
    DwtCompString title;
    int min_value, max_value;
    int decimal_points;
    int value;
    unsigned char orientation;
    DwtCallbackPtr callback, drag_callback, help_callback;

Widget DwtScaleCreate (parent_widget, name,
                override_arglist, override_argcount)
    Widget parent_widget;
    char *name;
    ArgList override_arglist;
    int override_argcount;
```

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of

	the parent window. This argument sets the DwtNy core widget attribute.
width	Specifies the width of the widget window. (The window width is calculated based on the scale width, the label widths, and orientation.) This argument sets the DwtNwidth core widget attribute.
height	Specifies the height of the widget window. (The window height is calculated based on the scale height, the labels, and orientation.) This argument sets the DwtNheight core widget attribute.
scale_width	Specifies the width of the scale, excluding the scale labels. This argument sets the DwtNscaleWidth attribute associated with DwtScaleCreate.
scale_height	Specifies the height of the scale, excluding the scale labels. This argument sets the DwtNscaleHeight attribute associated with DwtScaleCreate.
title	Specifies the title text string to appear in the scale window widget. This argument sets the DwtNtitle attribute associated with DwtScaleCreate.
min_value	Specifies the value represented by the top or left end of the scale. This argument sets the DwtNminValue attribute associated with DwtScaleCreate.
max_value	Specifies the value represented by the bottom or right end of the scale. This argument sets the DwtNmaxValue attribute associated with DwtScaleCreate.
decimal_points	Specifies the number of decimal points to shift the current slider value for display of the next slider position. This argument sets the DwtNdecimalPoints attribute associated with DwtScaleCreate.
value	Specifies the current slider position along the scale (the value selected by the user). This argument sets the DwtNvalue attribute associated with DwtScaleCreate.
orientation	Specifies whether the scale is displayed vertically or horizontally. You can pass DwtOrientationHorizontal or DwtOrientationVertical. This argument sets the DwtNorientation attribute associated with DwtScaleCreate.

callback Specifies the callback function or functions called back when the value of the scale changes. This argument sets the `DwtNvalueChangedCallback` attribute associated with `DwtScaleCreate`.

drag_callback Specifies the callback function or functions called when the user is dragging the scale slider. For this callback, the reason is `DwtCRDrag`. This argument sets the `DwtNdragCallback` attribute associated with `DwtScaleCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtScale` and `DwtScaleCreate` functions create an instance of the scale widget and return its associated widget ID. The scale widget is a primitive widget figure that allows the application to display a scale for vernier control of a specific parameter by the user. The user moves or drags a slider, which is part of the scale widget, and places the slider at a position representing the desired value. The scale may have labeled text at any number of points identifying the values corresponding to the points. The scale can be made insensitive and used as an output value indicator only (for example, a thermometer or percent completion indicator).

The application passes lower and upper values for the scale as integers and can (optionally) indicate a decimal point position. For example, a `DwtNminValue` of 100, a `DwtNmaxValue` of 10000, and a `DwtNdecimalPoints` of 2 would produce a scale from 1.00 to 100.00. Possible values returned from this example could be 230 or 5783.

Scale widget labels are provided by its children. The labels can be any widgets created using the scale widget as the parent.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Calculated based on scale width, the label widths, and the orientation
DwtNheight	Dimension	Calculated based on scale height, the label widths, and the orientation
DwtNborderWidth	Dimension	zero pixels
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL
Common Attributes		
DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNvalue	int	zero
DwtNtitle	DwtCompString	Scale name
DwtNorientation	unsigned char	DwtOrientationHorizontal
DwtNscaleWidth	Dimension	100 pixels
DwtNscaleHeight	Dimension	20 pixels
DwtNminValue	int	Zero
DwtNmaxValue	int	100
DwtNdecimalPoints	short	Zero
DwtNshowValue	Boolean	True
DwtNdragCallback	DwtCallbackPtr	NULL
DwtNvalueChangedCallback	DwtCallbackPtr	NULL

DwtNvalue
: Specifies the current slider position along the scale (the value selected by the user).

DwtNtitleType
: Specifies the title type. You can pass DwtCString or DwtPixmap.

DwtNtitle
: Specifies the title text string to appear in the scale window widget.

DwtNorientation
: Specifies whether the scale is displayed vertically or horizontally. You can pass DwtOrientationHorizontal or DwtOrientationVertical.

DwtNscaleWidth
: Specifies the thickness in pixels of the scale itself, not counting the labels.

DwtNscaleHeight
: Specifies the height of the scale, excluding the scale labels.

DwtNminValue
: Specifies the value represented by the top or left end of the scale.

DwtNmaxValue
: Specifies the value represented by the bottom or right end of the scale.

DwtNdecimalPoints
: Specifies the number of decimal points to shift the current slider value for display of the next slider position.

DwtNshowValue — Specifies a boolean value that, when True, states that the current value of the slider label string will be displayed next to the slider.

DwtNdragCallback — Specifies the callback function or functions called when the user is dragging the scale slider. For this callback, the reason is DwtCRDrag.

DwtNvalueChangedCallback — Specifies the callback function or functions called when the scale value was changed. For this callback, the reason is DwtCRValueChanged.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        int value;
} DwtScaleCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRValueChanged — The user moved the slider in the scale with drag or click.

DwtCRDrag — The user is dragging the slider.

DwtCRHelpRequested — The user selected Help.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*.

The value member is set to the current value of the scale.

See Also

DwtScaleGetSlider (3Dwt), DwtScaleSetSlider (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtScaleGetSlider (3Dwt)

Name

DwtScaleGetSlider – Gets the current value of the slider position displayed in the scale.

Syntax

void DwtScaleGetSlider(*widget*, *value_return*)
Widget *widget*;
int **value_return*;

Arguments

widget	Specifies the scale widget ID.
value_return	Returns the current slider position value.

Description

The `DwtScaleGetSlider` function returns the current slider position value displayed in the scale for the application.

See Also

DwtScaleSetSlider (3Dwt), DwtScale (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtScaleSetSlider – Sets or changes the current value of the slider position displayed in the scale.

Syntax

void DwtScaleSetSlider(*widget*, *value*)
Widget *widget*;
int *value*;

Arguments

widget	Specifies the scale widget ID.
value	Specifies the current slider position along the scale (the value selected by the user). This argument sets the `DwtNvalue` attribute associated with `DwtScaleCreate`.

Description

The `DwtScaleSetSlider` function sets or changes the current slider position value within the scale widget display for the application.

See Also

DwtScaleGetSlider (3Dwt), DwtScale (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtScrollBar (3Dwt)

Name

DwtScrollBar, DwtScrollBarCreate – Creates a scroll bar widget for the application to display and process scroll bar screen operations.

Syntax

```
Widget DwtScrollBar(parent_widget, name, x, y,
                    width, height, inc, page_inc,
                    shown, value, min_value, max_value,
                    orientation, callback, help_callback,
                    unit_inc_callback, unit_dec_callback,
                    page_inc_callback, page_dec_callback,
                    to_top_callback, to_bottom_callback)
                    drag_callback)
    Widget parent_widget;
    char *name;
    Position x, y;
    Dimension width, height;
    int inc, page_inc;
    int shown;
    int value;
    int min_value, max_value;
    int orientation;
    DwtCallbackPtr callback, help_callback;
    DwtCallbackPtr unit_inc_callback, unit_dec_callback;
    DwtCallbackPtr page_inc_callback, page_dec_callback;
    DwtCallbackPtr to_top_callback, to_bottom_callback;
    DwtCallbackPtr drag_callback;

Widget DwtScrollBarCreate (parent_widget, name,
                           override_arglist, override_argcount)
    Widget parent_widget;
    char *name;
    ArgList override_arglist;
    int override_argcount;
```

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the

	widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNx core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the DwtNy core widget attribute.
width	Specifies the width of the widget window. This argument sets the DwtNwidth core widget attribute.
height	Specifies the height of the widget window. This argument sets the DwtNheight core widget attribute.
inc	Specifies the amount of button increment and decrement. If this argument is nonzero, the scroll bar widget automatically adjusts the slider when an increment or decrement action occurs. This argument sets the DwtNinc attribute associated with DwtScrollBarCreate.
page_inc	Specifies the amount of page increment and decrement. If this argument is nonzero, the scroll bar widget automatically adjusts the slider when an increment or decrement action occurs. This argument sets the DwtNpageInc attribute associated with DwtScrollBarCreate.
shown	Specifies the size of the slider as a value between zero and the absolute value of DwtNmaxValue minus DwtNminValue. The size of the slider varies, depending on how much of the slider scroll area it represents. This argument sets the DwtNshown attribute associated with DwtScrollBarCreate.
value	Specifies the scroll bar's top thumb position between DwtNminValue and DwtNmaxValue. This sets the DwtNvalue attribute associated with DwtScrollBarCreate.
min_value	Specifies the scroll bar's minimum value. This argument sets the DwtNminValue attribute associated with DwtScrollBarCreate.
max_value	Specifies the scroll bar's maximum value. This argument sets the DwtNmaxValue attribute associated with DwtScrollBarCreate.
orientation	Specifies whether the scroll bar is displayed vertically or

horizontally. You can pass DwtOrientationHorizontal or DwtOrientationVertical. This argument sets the DwtNorientation attribute associated with DwtScrollBarCreate.

callback Specifies the callback function or functions called back when the value of the scroll bar changes. This argument sets the DwtNvalueChangedCallback attribute associated with DwtScrollBarCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

unit_inc_callback
Specifies the callback function or functions called when the user selected the down or right unit scroll function. For this callback, the reason is DwtCRUnitInc. This argument sets the DwtNunitIncCallback attribute associated with DwtScrollBarCreate.

unit_dec_callback
Specifies the callback function or functions called when the user selected the above or left unit scroll function. For this callback, the reason is DwtCRUnitDec. This argument sets the DwtNunitDecCallback attribute associated with DwtScrollBarCreate.

page_inc_callback
Specifies the callback function or functions called when the user selected the below or right page scroll function. For this callback, the reason is DwtCRPageInc. This argument sets the DwtNpageIncCallback attribute associated with DwtScrollBarCreate.

page_dec_callback
Specifies the callback function or functions called when the user selected the above or left page scroll function. For this callback, the reason is DwtCRPageDec. This argument sets the DwtNpageDecCallback attribute associated with DwtScrollBarCreate.

to_top_callback
Specifies the callback function or functions called when the user selected the current line to top scroll function. For this

callback, the reason is DwtCRToTop. The scroll bar does not automatically change the scroll bar's DwtNvalue for this callback. This argument sets the DwtNtoTopCallback attribute associated with DwtScrollBarCreate.

to_bottom_callback
Specifies the callback function or functions called when the user selected the current line to bottom scroll function. For this callback, the reason is DwtCRToBottom. The scroll bar does not automatically change the scroll bar's DwtNvalue for this callback. This argument sets the DwtNtoBottomCallback attribute associated with DwtScrollBarCreate.

drag_callback Specifies the callback function or functions called when the user is dragging the scroll bar slider. For this callback, the reason is DwtCRDrag. This argument sets the DwtNdragCallback attribute associated with DwtScrollBarCreate.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtScrollBar and DwtScrollBarCreate functions create an instance of a scroll bar widget and return its associated widget ID. The scroll bar widget is a screen object that the application or user uses to scroll through display data too large for the screen. This widget consists of two stepping arrows at either end of an elongated rectangle called the scroll region. The scroll region is overlaid with a slider bar (thumb) that is adjusted in size and position (thumb shown) as scrolling occurs using the function attributes. The stepping arrows and the exposed scroll areas behind the slider are the scroll activator objects providing the user interface syntax "feel."

If the default core widget attributes DwtNwidth or DwtNheight (0) are used, the scroll bar is set to the DwtNheight of the parent window (vertical) or to the DwtNwidth of the parent window (horizontal). If the default core widget attributes DwtNx or DwtNy (0) are used, the scroll bar is set to the right of the parent window (vertical) or to the bottom of the

parent window (horizontal). This is also true if you specify DwtNwidth, DwtNheight, DwtNx, or DwtNy in the call to XtSetValues.

Note that the DwtNtoTopCallback and DwtNtoBottomCallback callbacks do not automatically set the thumb as the other callbacks do.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	For vertical scroll bars, 17 pixels. For horizontal scroll bars, the width of the parent minus 17 pixels.
DwtNheight	Dimension	For horizontal scroll bars, 17 pixels. For vertical scroll bars, the height of the parent minus 17 pixels.
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNvalue	int	Zero
DwtNminValue	int	Zero
DwtNmaxValue	int	100
DwtNorientation	unsigned char	DwtOrientationVertical
DwtNtranslations1	XtTranslations	NULL
DwtNtranslations2	XtTranslations	NULL
DwtNshown	int	10 units
DwtNinc	int	10 units
DwtNpageInc	int	10 units
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNunitIncCallback	DwtCallbackPtr	NULL
DwtNunitDecCallback	DwtCallbackPtr	NULL
DwtNpageIncCallback	DwtCallbackPtr	NULL
DwtNpageDecCallback	DwtCallbackPtr	NULL
DwtNtoTopCallback	DwtCallbackPtr	NULL
DwtNtoBottomCallback	DwtCallbackPtr	NULL
DwtNdragCallback	DwtCallbackPtr	NULL
DwtNshowArrows	Boolean	True

DwtNvalue — Specifies the scroll bar's top thumb position between DwtNminValue and DwtNmaxValue. This attribute also appears as a member in DwtScrollBarCallbackStruct.

DwtNminValue — Specifies the scroll bar's minimum value.

DwtNmaxValue — Specifies the scroll bar's maximum value.

DwtNorientation — Specifies whether the scroll bar is displayed vertically or horizontally. You can pass

DwtOrientationHorizontal or DwtOrientationVertical.

DwtNtranslations1
: Specifies the translation table for events after being parsed by the X intrinsics function XtParseTranslationTable for the decrement button.

DwtNtranslations2
: Specifies the translation table for events after being parsed by the X intrinsics function XtParseTranslationTable for the increment button.

DwtNshown
: Specifies the size of the slider as a value between zero and the absolute value of DwtNmaxValue minus DwtNminValue. The size of the slider varies, depending on how much of the slider scroll area it represents.

DwtNinc
: Specifies the amount of button increment and decrement. If this argument is nonzero, the scroll bar widget automatically adjusts the slider when an increment or decrement action occurs.

DwtNpageInc
: Specifies the amount of page increment and decrement. If this argument is nonzero, the scroll bar widget automatically adjusts the slider when an increment or decrement action occurs.

DwtNvalueChangedCallback
: Specifies the callback function or functions called when the value of the scroll bar slider was changed. For this callback, the reason is DwtCRValueChanged.

DwtNunitIncCallback
: Specifies the callback function or functions called when the user selected the down or right unit scroll function. For this callback, the reason is DwtCRUnitInc.

`DwtNunitDecCallback`
: Specifies the callback function or functions called when the user selected the above or left unit scroll function. For this callback, the reason is `DwtCRUnitDec`.

`DwtNpageIncCallback`
: Specifies the callback function or functions called when the user selected the below or right page scroll function. For this callback, the reason is `DwtCRPageInc`.

`DwtNpageDecCallback`
: Specifies the callback function or functions called when the user selected the above or left page scroll function. For this callback, the reason is `DwtCRPageDec`.

`DwtNtoTopCallback`
: Specifies the callback function or functions called when the user selected the current line to top scroll function. For this callback, the reason is `DwtCRToTop`. The scroll bar does not automatically change the scroll bar's `DwtNvalue` for this callback.

`DwtNtoBottomCallback`
: Specifies the callback function or functions called when the user selected the current line to bottom scroll function. For this callback, the reason is `DwtCRToBottom`. The scroll bar does not automatically change the scroll bar's `DwtNvalue` for this callback.

`DwtNdragCallback`
: Specifies the callback function or functions called when the user is dragging the scroll bar slider. For this callback, the reason is `DwtCRDrag`. The scroll bar does not automatically change the scroll bar's `DwtNvalue` for this callback.

`DwtNshowArrows`
: Specifies a boolean value that, when `True`, indicates there are arrows. If `False`, there are no arrows.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        int value;
                        int pixel;
} DwtScrollBarCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRValueChanged`	The user changed the value of the scroll bar slider.
`DwtCRUnitInc`	The user selected the down or right unit scroll function.
`DwtCRUnitDec`	The user selected the up or left unit scroll function.
`DwtCRPageDec`	The user selected the above or left page scroll function.
`DwtCRPageInc`	The user selected the below or right page scroll function.
`DwtCRToTop`	The user selected the current line to top scroll function.
`DwtCRToBottom`	The user selected the current line to bottom scroll function.
`DwtCRDrag`	The user is dragging the scroll bar slider.
`DwtCRHelpRequested`	The user selected help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on

XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*. The value member is set to the slider's current value and maps to the DwtNvalue attribute. The pixel member is set to the pixel value from the top right of the scroll bar where the event occurred. This pixel value is used for the DwtNtoTopCallback and DwtNtoBottomCallback attributes.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtScrollBarGetSlider (3Dwt)

Name

DwtScrollBarGetSlider – Retrieves the current size and position parameters of the slider in the scroll bar widget.

Syntax

void DwtScrollBarGetSlider(*widget*, *value_return*, *shown_return*, *inc_return*, *pageinc_return*)
Widget *widget*;
int **value_return*;
int **shown_return*;
int **inc_return*;
int **pageinc_return*;

Arguments

widget	Specifies the scroll bar widget ID.
value_return	Returns the scroll bar's top thumb (slider) position between the `DwtNminValue` and `DwtNmaxValue` attributes to the scroll bar widget.
shown_return	Returns the size of the slider as a value between zero and the absolute value of `DwtNmaxValue` minus `DwtNminValue`. The size of the slider varies, depending on how much of the slider scroll area it represents.
inc_return	Returns the amount of button increment and decrement.
pageinc_return	Returns the amount of page increment and decrement.

Description

The `DwtScrollBarGetSlider` function returns the currently displayed size/position values of the slider in the scroll bar widget. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main scroll bar or set slider function attributes. The stepping arrows and the slider are the scroll activator objects providing the user interface syntax "feel."

See Also

DwtScrollBarSetSlider (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtScrollBarSetSlider – Sets or changes the current size/position parameters of the slider in the scroll bar widget.

Syntax

void DwtScrollBarSetSlider(*widget*, *value*, *shown*, *inc*,
page_inc, *notify*)
Widget *widget*;
int *value*;
int *shown*;
int *inc*, *page_inc*;
Boolean *notify*;

Arguments

widget	Specifies the scroll bar widget ID.
value	Specifies the scroll bar's top thumb (slider) position between `DwtNminValue` and `DwtNmaxValue`. The attribute name associated with this argument is `DwtNvalue`.
shown	Specifies the size of the slider as a value between zero and the absolute value of `DwtNmaxValue` minus `DwtNminValue`. The size of the slider varies, depending on how much of the slider scroll area it represents. This argument sets the `DwtNshown` attribute associated with `DwtScrollBarCreate`.
inc	Specifies the amount of button increment and decrement. If this argument is nonzero, the scroll bar widget automatically adjusts the slider when an increment or decrement action occurs. This argument sets the `DwtNinc` attribute associated with `DwtScrollBarCreate`.
page_inc	Specifies the amount of page increment and decrement. If this argument is nonzero, the scroll bar widget automatically adjusts the slider when an increment or decrement action occurs. This argument sets the `DwtNpageInc` attribute associated with `DwtScrollBarCreate`.
notify	Specifies a boolean value that, when `True`, indicates a change in the scroll bar value and that the scroll bar widget automatically activates the

DwtNvalueChangedCallback with the recent change. If False, no change in the scroll bar's value has occurred and DwtNvalueChangedCallback is not activated.

Description

The DwtScrollBarSetSlider function sets or changes the currently displayed scroll bar widget slider for the application. The scroll region is overlaid with a slider bar that is adjusted in size and position using the main scroll bar or set slider function attributes. The stepping arrows and the slider are the scroll activator objects providing the user interface syntax "feel."

See Also

DwtScrollBarGetSlider (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtScrollWindow, DwtScrollWindowCreate – Creates a scroll window widget for simple applications in the main window widget work area.

Syntax

Widget DwtScrollWindow(*parent_widget*, *name*, *x*, *y*,
width, *height*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
Dimension *width*, *height*;

Widget DwtScrollWindowCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the top of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
width	Specifies in pixels the width of the widget window. This argument sets the `DwtNwidth` core widget attribute.
height	Specifies in pixels the height of the widget window. This argument sets the `DwtNheight` core widget attribute.
override_arglist	Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtScrollWindow` and `DwtScrollWindowCreate` functions create an instance of a scroll window widget and return its associated widget ID. This widget provides a more direct XUI interface for applications with scroll bars. When calling `DwtScrollWindow`, you set the scroll window widget attributes presented in the formal parameter list. For `DwtScrollWindowCreate`, you specify a list of attribute name/value pairs that represent all the possible scroll window widget attributes.

The `DwtScrollWindow` and `DwtScrollWindowCreate` functions create a composite widget that can contain vertical and horizontal scroll bar widgets and any widget as the window region. Scroll bar positioning and scroll bar slider sizes are automatically maintained. The scroll window widget simplifies programming by allowing you to create an application with scroll bars directly in the scroll window widget work area.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	Widget-specific
`DwtNheight`	`Dimension`	Widget-specific
`DwtNborderWidth`	`Dimension`	One pixel
`DwtNborder`	`Pixel`	Default foreground color
`DwtNborderPixmap`	`Pixmap`	NULL
`DwtNbackground`	`Pixel`	Default background color
`DwtNbackgroundPixmap`	`Pixmap`	NULL
`DwtNcolormap`	`Colormap`	Default color map

DwtNsensitive	Boolean	True Setting the sensitivity of the scroll window causes all widgets contained in that window to be set to the same sensitivity as the scroll window.
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	NOT SUPPORTED	

Widget-Specific Attributes

You can set the following widget-specifc attributes in the *override_arglist*:

Attribute Name	Data Type	Default
DwtNhorizontalScrollBar	Widget	NULL
DwtNverticalScrollBar	Widget	NULL
DwtNworkWindow	Widget	NULL
DwtNshownValueAutomaticHoriz	Boolean	True
DwtNshownValueAutomaticVert	Boolean	True

DwtNhorizontalScrollBar

Specifies the scroll bar widget ID for the horizontal scroll bar to be associated with the scroll window widget. You can set this ID only after creating an

instance of the main window widget.

`DwtNverticalScrollBar`
: Specifies the scroll bar widget ID for the vertical scroll bar to be associated with the scroll window widget. You can set this ID only after creating an instance of the main window widget.

`DwtNworkWindow`
: Specifies the widget ID for the work window to be associated with the scroll window widget. You can set this ID only after creating an instance of the main window widget.

`DwtNshownValueAutomaticHoriz`
: Specifies a boolean value that, when `True`, indicates that `DwtScrollWindow` automatically sets the value for the `DwtNshown` attribute for the specified horizontal scroll bar widget.

`DwtNshownValueAutomaticVert`
: Specifies a boolean value that, when `True`, indicates that `DwtScrollWindow` automatically sets the value for the `DwtNshown` attribute for the specified vertical scroll bar widget.

Return Value

These functions return the ID of the created widget.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtScrollWindowSetAreas – Sets up or adds the window region, and the horizontal or vertical scroll bar widgets to the scroll window widget.

Syntax

void DwtScrollWindowSetAreas(*widget*, *horizontal_scroll_bar*, *vertical_scroll_bar*, *work_region*)
Widget *widget*;
Widget *horizontal_scroll_bar*;
Widget *vertical_scroll_bar*;
Widget *work_region*;

Arguments

widget Specifies the scroll window widget ID.

horizontal_scroll_bar
Specifies the scroll bar widget ID for the horizontal scroll bar to be associated with the scroll window widget. You can set or specify this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNhorizontalScrollBar`.

vertical_scroll_bar
Specifies the scroll bar widget ID for the vertical scroll bar to be associated with the scroll window widget. You can set or specify this ID only after creating an instance of the main window widget. The attribute name associated with this argument is `DwtNverticalScrollBar`.

work_region Specifies the widget ID for the window to be associated with the scroll window work area. You can set or specify this ID only after you create an instance of the main window widget.

Description

The `DwtScrollWindowSetAreas` function adds or changes a window work region and a horizontal or vertical scroll bar widget to the scroll window widget for the application. You must call this function before the scroll window widget is realized, that is, before calling the X intrinsics function `XtRealizeWidget`. Each widget is optional and may be passed as NULL.

See Also

DwtScrollWindow (3Dwt), DwtWindow (3Dwt),
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSelection, DwtSelectionCreate – Creates a selection box widget.

Syntax

```
Widget DwtSelection(parent_widget, name, x, y,
                    title, value, items,
                    item_count, visible_items_count, style,
                    default_position, callback, help_callback)
    Widget parent_widget;
    char *name;
    Position  x, y;
    DwtCompString title;
    DwtCompString value;
    DwtCompString *items;
    int item_count, visible_items_count;
    int style;
    Boolean default_position;
    DwtCallbackPtr callback, help_callback;

Widget DwtSelectionCreate (parent_widget, name,
                           override_arglist, override_argcount)
    Widget parent_widget;
    char *name;
    ArgList override_arglist;
    int override_argcount;
```

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
title	Specifies the text that appears in the banner of the selection

box. This argument sets the DwtNtitle attribute associated with DwtDialogBoxCreate.

value Specifies the text in the text edit field. This argument sets the DwtNvalue attribute associated with DwtSelectionCreate.

items Specifies the items in the selection widget's list box. This argument sets the DwtNitems attribute associated with DwtSelectionCreate.

item_count Specifies the number of items in the selection widget's list box. This argument sets the DwtNitemsCount associated with DwtSelectionCreate.

visible_items_count
Specifies the number of items displayed in the selection widget's list box. This argument sets the DwtNvisibleItemsCount attribute associated with DwtSelectionCreate.

style Specifies the style of the pop-up dialog box widget. You can pass DwtModal (modal) or DwtModeless (modeless). This argument sets the DwtNstyle attribute associated with DwtDialogBoxPopupCreate.

default_position
Specifies a boolean value that, when True, causes DwtNx and DwtNy to be ignored and forces the default widget position. The default widget position is centered in the parent window. If False, the specified DwtNx and DwtNy attributes are used to position the widget. This argument sets the DwtNdefaultPosition attribute associated with DwtDialogBoxCreate.

callback Specifies the callback function or functions called when the user makes or cancels a selection, or there is no match for the item selected by the user. This argument sets the DwtNactivateCallback, DwtNcancelCallback, and DwtNnoMatchCallback attributes associated with DwtSelectionCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

override_arglist

Specifies the application override argument list.

override_argcount

Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtSelection` and `DwtSelectionCreate` functions create an instance of a selection box widget and return its associated widget ID.

When calling `DwtSelection`, you set the selection box widget attributes presented in the formal parameter list. For `DwtSelectionCreate`, however, you specify a list of attribute name/value pairs that represent all the possible selection box widget attributes. The selection widget is a pop-up dialog box containing a label widget, a text entry widget holding the current value, a list box displaying the current item list, and Ok and Cancel push buttons.

When realized, the selection widget displays the item list passed by the caller. The current value is displayed in the text entry field. Users make selections by clicking the mouse in the list box or by typing item names in the text entry field. The selection widget does not do file searches. To perform file searches, use `DwtFileSelectionCreate`.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Centered in the parent window
`DwtNy`	`Position`	Centered in the parent window
`DwtNwidth`	`Dimension`	The width of the list box, plus the width of the push buttons, plus three times `DwtNmarginWidth`. The list box will grow to accommodate items wider than the title.
`DwtNheight`	`Dimension`	The height of the list box, plus the height of the text edit field, plus the height of the label, plus three times `DwtNmarginHeight`.
`DwtNborderWidth`	`Dimension`	One pixel

DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Dialog Pop-Up Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNunits	unsigned char	DwtFontUnits
DwtNstyle	unsigned char	DwtModal
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	XtTranslations	NULL
DwtNmarginWidth	Dimension	5 pixels
DwtNmarginHeight	Dimension	5 pixels
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeGrowOnly
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNtitle	DwtCompString	"Open"
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNtakeFocus	Boolean	True for modal dialog box

		`False` for modeless dialog box
`DwtNautoUnmanage`	`Boolean`	`True`
`DwtNdefaultButton`	`Widget`	NULL
`DwtNcancelButton`	`Widget`	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
`DwtNlabel`	`DwtCompString`	"Items"
`DwtNvalue`	`DwtCompString`	""
`DwtNokLabel`	`DwtCompString`	"Ok"
`DwtNcancelLabel`	`DwtCompString`	"Cancel"
`DwtNactivateCallback`	`DwtCallbackPtr`	NULL
`DwtNcancelCallback`	`DwtCallbackPtr`	NULL
`DwtNnoMatchCallback`	`DwtCallbackPtr`	NULL
`DwtNvisibleItemsCount`	int	8
`DwtNitems`	`DwtCompString *`	NULL
`DwtNitemsCount`	int	Zero
`DwtNmustMatch`	`Boolean`	`False`
`DwtNselectionLabel`	`DwtCompString`	"Selection"

`DwtNlabel` Specifies the label to appear above the list box containing the items.

`DwtNvalue` Specifies the text in the text edit field.

`DwtNselectionLabel`
Specifies the label above the selection text entry field.

`DwtNokLabel` Specifies the label for the Ok push button. If the label is a NULL string, the button is not displayed.

`DwtNcancelLabel` Specifies the label for the Cancel push button. If the label is a NULL string, the button is not displayed.

`DwtNactivateCallback`
Specifies the callback function or functions called when the user makes a selection. For this callback, the reason is `DwtCRActivate`.

`DwtNcancelCallback`
Specifies the callback function or functions called when the user clicks on the Cancel button. For this

callback, the reason is DwtCRCancel.

DwtNnoMatchCallback
: Specifies the callback function or functions called when the user's selection does not have an exact match with any items in the list box. This callback is activated only if DwtNmustMatch is True. For this callback, the reason is DwtCRNoMatch.

DwtNvisibleItemsCount
: Specifies the number of items displayed in the selection widget's list box.

DwtNitems
: Specifies the items in the selection widget's list box.

DwtNitemsCount
: Specifies the number of items in the selection widget's list box.

DwtNmustMatch
: Specifies a boolean value that, when True, indicates that the selection widget checks whether the user's selection has an exact match in the list box. If the selection does not have an exact match, the DwtNnoMatchCallback is activated. If the selection has an exact match, the DwtNactivateCallback is activated.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
                    DwtCompString value;
                    int value_len;
} DwtSelectionCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRActivate
: The user activated the Ok push button or double clicked on an item that has an exact match in the list box.

DwtCRNoMatch	The user activated the Ok push button or double clicked on an item that does not have an exact match in the list box.
DwtCRCancel	The user activated the Cancel button.
DwtCRHelpRequested	The user selected help somewhere in the file selection box.

The event member is a pointer to the Xlib structure XEvent, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on XEvent and event processing, see the *Guide to the Xlib Library: C Language Binding*. The value member is set to the current selection when the callback occurred. The value_len member is set to the length of the selection compound-string.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSeparator (3Dwt)

Name

DwtSeparator, DwtSeparatorCreate – Creates a separator widget for the application to define a border between items in a display.

Syntax

Widget DwtSeparator(*parent_widget*, *name*, *x*, *y*, *orientation*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
unsigned char *orientation*;

Widget DwtSeparatorCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
orientation	Specifies whether the separator is displayed vertically or horizontally. You can pass `DwtOrientationHorizontal` or `DwtOrientationVertical`. This argument sets the `DwtNorientation` attribute associated with `DwtSeparatorCreate`. A separator widget draws a centered single pixel line between the appropriate margins. For example, a horizontal separator draws a horizontal line from the left margin to the

right margin. It is placed vertically in the middle of the widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtSeparator` and `DwtSeparatorCreate` functions create an instance of the separator widget and return its associated widget ID. When calling `DwtSeparator`, you set the widget attributes presented in the formal parameter list. For `DwtSeparatorCreate`, however, you specify a list of attribute name/value pairs that represent all the possible separator widget attributes.

The separator widget is a screen object that allows the application to draw a separator between items in a display. The separator widget draws horizontal or vertical lines in inactive areas of a window (typically menus). Because a separator widget does not support children, it always refuses geometry requests. The separator widget does nothing on a resize by its parents.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	3 pixels
`DwtNheight`	`Dimension`	3 pixels
`DwtNborderWidth`	int	zero
`DwtNborder`	`Pixel`	Default foreground color
`DwtNborderPixmap`	`Pixmap`	NULL
`DwtNbackground`	`Pixel`	Default background color
`DwtNbackgroundPixmap`	`Pixmap`	NULL
`DwtNcolormap`	`Colormap`	Default color map
`DwtNsensitive`	`Boolean`	`True`

DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	NOT SUPPORTED	
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	NOT SUPPORTED	

Label Attributes

DwtNlabelType	unsigned char	DwtCString
DwtNlabel	DwtCompString	Widget name
DwtNmarginWidth	Dimension	Two pixels for text Zero pixels for pixmap
DwtNmarginHeight	Dimension	Two pixels for text Zero pixels for pixmap
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNpixmap	Pixmap	NULL
DwtNmarginLeft	Dimension	Zero
DwtNmarginRight	Dimension	Zero
DwtNmarginTop	Dimension	Zero
DwtNmarginBottom	Dimension	Zero
DwtNconformToText	Boolean	True, if the widget is created with a width and height of zero False, if the widget is created with a non-zero width and height

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNorientation	unsigned char	DwtOrientationHorizontal

DwtNorientation Specifies whether the separator is displayed vertically or horizontally. You can pass DwtOrientationHorizontal or DwtOrientationVertical. A separator widget draws a centered single pixel line between the appropriate margins. For example, a horizontal separator draws a horizontal line from the left margin to the right margin. It is placed vertically in the middle of the widget.

Return Value

These functions return the ID of the created widget.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSeparatorGadgetCreate (3Dwt)

Name

DwtSeparatorGadgetCreate – Creates a separator gadget.

Syntax

Widget DwtSeparatorGadgetCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)

Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtSeparatorGadgetCreate` function creates an instance of the separator gadget and returns its associated gadget ID. A separator gadget is similar in appearance and semantics to a separator widget. Like all gadgets, `DwtSeparatorGadgetCreate` does not have a window but uses the window of the closest antecedent widget. Thus, the antecedent widget provides all event dispatching for the gadget. This currently restricts gadgets to being descendents of menu or dialog class (or subclass) widgets.

Inherited Attributes

Attribute Name	Data Type	Default
Rectangle Attributes		
`DwtNx`	`Position`	Determined by the geometry manager

DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	3 pixels
DwtNheight	Dimension	3 pixels
DwtNborderWidth	Dimension	zero
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNorientation	unsigned char	DwtOrientationHorizontal

DwtNorientation Specifies whether the separator is displayed vertically or horizontally. You can pass DwtOrientationHorizontal or DwtOrientationVertical. A separator gadget draws a centered single pixel line between the appropriate margins. For example, a horizontal separator draws a horizontal line from the left margin to the right margin. It is placed vertically in the middle of the gadget.

Return Value

This function returns the ID of the created widget.

Callback Information

There is no callback for this gadget.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtStartCopyFromClipboard (3Dwt)

Name

DwtStartCopyFromClipboard – Indicates that the application is ready to start copying data from the clipboard and locks the clipboard.

Syntax

int DwtStartCopyFromClipboard(*display*, *window*, *time*)
Display **display*;
Window *window*;
Time *time*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
time	Specifies the timestamping of the event that triggered the copy.

Description

The `DwtStartCopyFromClipboard` function notifies the cut and paste functions that the application is ready to start copying data from the clipboard. `DwtStartCopyFromClipboard` locks the clipboard and remains locked until you call `DwtEndCopyFromClipboard`.

After calling `DwtStartCopyFromClipboard`, an application can make multiple calls to `DwtCopyFromClipboard` requesting data in one or several formats. You specify the format by setting the *format_name* argument to `DwtCopyFromClipboard`. Each call to `DwtCopyFromClipboard` in a specified format results in data being incrementally copied from the clipboard until all data with the specified format has been copied. When all data in a specified format has been successfully copied, `DwtCopyFromClipboard` returns `ClipboardSuccess`. When more data remains to be copied in the specified format, `DwtCopyFromClipboard` returns `ClipboardTruncate`. An application can copy data in as many formats

as desired before calling DwtEndCopyFromClipboard.

It is recommended that any calls to inquire routines needed by the application be made between the call to DwtStartCopyFromClipboard and the call to DwtEndCopyFromClipboard. That way, the application does not need to call DwtClipboardLock and DwtClipboardUnlock.

To perform cut and paste operations between your application and an application using the ICCCM clipboard selection mechanism, you must use DwtStartCopyToClipboard and provide a timestamping value for *time*, not a CurrentTime value. Use of the value CurrentTime for *time* may cause the ICCCM interface to fail.

Applications do not need to use DwtStartCopyFromClipboard and DwtEndCopyFromClipboard, in which case DwtCopyFromClipboard works as documented. However, using these two functions allows incremental copying from the clipboard and ensures ICCCM compatibility.

Return Value

This function returns one of these status return constants:

ClipboardSuccess	The function is successful.
ClipboardLocked	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtCopyFromClipboard (3Dwt), DwtEndCopyFromClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtStartCopyToClipboard (3Dwt)

Name

DwtStartCopyToClipboard – Sets up storage and data structures to receive clipboard data.

Syntax

int DwtStartCopyToClipboard(*display*, *window*, *clip_label*,
time, *widget*, *callback*, *item_id*)
Display **display*;
Window *window*;
DwtCompString *clip_label*;
Time *time*;
Widget *widget*;
VoidProc *callback*;
long **item_id*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.
clip_label	Specifies the label to be associated with the data item. This argument is used to identify the data item, for example, in a clipboard viewer. An example of a label is the name of the application that places the data in the clipboard.
time	Specifies the timestamping of the event that triggered the copy.
widget	Specifies the ID of the widget that will receive messages requesting data previously passed by name. This argument must be present in order to pass data by name. Any valid widget ID in your application can be used. All message handling is done by the cut and paste functions.
callback	Specifies the address of the callback function that is called when the clipboard needs data that was originally passed by

name. This is also the callback to receive the DELETE message for items that were originally passed by name. This argument must be present in order to pass data by name.

item_id Specifies the number assigned to this data item. The application uses this number in calls to `DwtCopyToClipboard`, `DwtEndCopyToClipboard`, and `DwtCancelCopyToClipboard`.

Description

The `DwtStartCopyToClipboard` function sets up storage and data structures to receive clipboard data. An application calls `DwtStartCopyToClipboard` during a cut or copy operation. The data item that these structures receive through calls to `DwtCopyToClipboard` then becomes the next item to be pasted (the next-paste item) in the clipboard after the call to `DwtEndCopyToClipboard`.

`DwtStartCopyToClipboard` is like `DwtBeginCopyToClipboard` except that it has the *time* argument to support the ICCCM clipboard selection mechanism. To perform cut and paste operations between your application and an application using the ICCCM clipboard selection mechanism, you must use `DwtStartCopyToClipboard` and provide a timestamping value for *time*, not a `CurrentTime` value. Use of the value `CurrentTime` for *time* may cause the ICCCM interface to fail.

The *window* and *callback* arguments must be present in order to pass data by name.

The callback format is as follows:

function name(*widget*, *data_id*, *private_id*, *reason*)
    Widget **widget*;
    int **data_id*;
    int **private_id*;
    int **reason*;

widget Specifies the ID of the widget passed to `DwtStartCopyToClipboard`.

data_id Specifies the identifying number returned by `DwtCopyToClipboard`, which identifes the pass-by-name data.

private_id Specifies the private information passed to `DwtCopyToClipboard`.

reason Specifies the reason, which is either

`DwtCRClipboardDataDelete` or
`DwtCRClipboardDataRequest`.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

DwtCopyToClipboard (3Dwt), DwtEndCopyToClipboard (3Dwt), DwtCancelCopyToClipboard (3Dwt), DwtBeginCopyToClipboard (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSText, DwtSTextCreate – Creates a simple text widget for the application to display a single or multiline text field. The user can enter and edit text in this field.

Syntax

Widget DwtSText(*parent_widget*, *name*, *x*, *y*, *cols*, *rows*, *value*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
Dimension *cols*, *rows*;
char **value*;

Widget DwtSTextCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
cols	Specifies the width of the text window measured in character spaces. 888 This argument sets the `DwtNcols` attribute associated with `DwtSTextCreate`.
rows	Specifies the height of the text window measured in character heights or number of line spaces. This argument sets the `DwtNrows` attribute associated with `DwtSTextCreate`.
value	Specifies the actual text to display. This argument sets the

DwtNvalue attribute associated with DwtSTextCreate.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtSText and DwtSTextCreate functions create an instance of a simple text widget and return its associated widget ID. When calling DwtSText, you set the text widget attributes presented in the formal parameter list. For DwtSTextCreate, however, you specify a list of attribute name/value pairs that represent all the possible simple text widget attributes.

The text widget enables the application to display a single or multiline field of text for input and edit manipulation by the user. By default, the text window grows or shrinks as the user enters or deletes text characters. Note that the text window does not shrink below the initial size set at creation time.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Set as large as necessary to display the DwtNrows with the specified DwtNmarginWidth
DwtNheight	Dimension	As large as necessary to display the DwtNcols with the specified DwtNmarginHeight
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color

DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

You can set the following widget-specifc attributes in the *override_arglist*:

Attribute Name	Data Type	Default
DwtNmarginWidth	Dimension	2 pixels
DwtNmarginHeight	Dimension	Two pixels
DwtNcols	Dimension	20 characters
DwtNrows	Dimension	1 character
DwtNtopPosition	DwtTextPosition	Zero
DwtNwordWrap	Boolean	False
DwtNscrollVertical	Boolean	False
DwtNresizeHeight	Boolean	True
DwtNresizeWidth	Boolean	True
DwtNvalue	char *	""
DwtNeditable	Boolean	True
DwtNmaxLength	int	2**31-1
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNlostFocusCallback	DwtCallbackPtr	NULL
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNinsertionPointVisible	Boolean	True
DwtNautoShowInsertPoint	Boolean	True
DwtNinsertionPosition	int	Zero
DwtNforeground	Pixel	The current server's default foreground
DwtNfont	DwtFontList	The current server font list.
DwtNblinkRate	int	500 milliseconds

DwtNscrollLeftSide	Boolean	False
DwtNhalfBorder	Boolean	True
DwtNpendingDelete	Boolean	True
DwtNuserData	Opaque *	NULL

DwtNmarginWidth
Specifies the number of pixels between the left or right edge of the window and the text.

DwtNmarginHeight
Specifies the number of pixels between the top or bottom edge of the window and the text.

DwtNcols
Specifies the width of the text window measured in character spaces.

DwtNrows
Specifies the height of the text window measured in character heights or number of line spaces.

DwtNtopPosition
Specifies the position to display at the top of the window.

DwtNwordWrap
Specifies a boolean value that, when True, indicates that lines are broken at word breaks and text does not run off the right edge of the window.

DwtNscrollVertical
Specifies a boolean value that, when True, adds a scroll bar that allows the user to scroll vertically through the text.

DwtNresizeHeight
Specifies a boolean value that, when True, indicates that the simple text widget will attempt to resize its height to accommodate all the text contained in the widget. If this is set to True, the text will always be displayed starting from the first position in the source, even if instructed otherwise. This attribute is ignored if DwtNscrollVertical is True.

DwtNresizeWidth
Specifies a boolean value that, when True, indicates that the simple text widget will attempt to resize its width to accommodate all the text contained in the widget. This argument is ignored if DwtNwordWrap is True.

DwtNvalue
Specifies the actual text to display.

DwtNeditable	Specifies a boolean value that, when True, indicates that the user can edit the text string in the simple text widget. If False, prohibits the user from editing the text string.
DwtNmaxLength	Specifies the maximum length of the text string in the simple text widget.
DwtNfocusCallback	Specifies the callback function or functions called when the simple text widget accepted the input focus. For this callback, the reason is DwtCRFocus.
DwtNhelpCallback	Specifies the callback function or functions called when a help request is made.
DwtNlostFocusCallback	Specifies the callback function or functions called when the simple text widget loses focus. For this callback, the reason is DwtCRLostFocus.
DwtNvalueChangedCallback	Specifies the callback function or functions called when the simple text widget value changed. For this callback, the reason is DwtCRValueChanged.
DwtNinsertionPointVisible	Specifies a boolean value that, when True, indicates that the insertion point is marked by a blinking text cursor.
DwtNautoShowInsertPoint	Specifies a boolean value that, when True, ensures that the text visible in the simple text widget window will contain the insertion point. This means that if the insertion point changes, the contents of the simple text widget window may scroll in order to bring the insertion point into the window.
DwtNinsertionPosition	Specifies the current location of the insertion point.
DwtNforeground	Specifies the pixel for the foreground of the simple text widget.
DwtNfont	Specifies the font list to be used for the simple text widget.

DwtNblinkRate	Specifies the blink rate of the text cursor in milliseconds.
DwtNscrollLeftSide	Specifies a boolean value that, when True, indicates that the vertical scroll bar should be placed on the left side of the simple text window. This attribute is ignored if DwtNscrollVertical is False.
DwtNhalfBorder	Specifies a boolean value that, when True, indicates that a border is displayed only on the left and bottom edges of the simple text widget.
DwtNpendingDelete	Specifies a boolean value that, when True, indicates that selected text containing the insertion point is deleted when new text is entered.
DwtNuserData	Specifies any user private data to be associated with the widget. The XUI Toolkit does not interpret this data.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

DwtCRFocus	The simple text widget has received the input focus.
DwtCRLostFocus	The simple text widget has lost the input focus.
DwtCRValueChanged	The user changed the value of the text string in the simple text widget.

`DwtCRHelpRequested` The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

DwtSTextGetString (3Dwt), DwtSTextSetString (3Dwt), DwtSTextReplace (3Dwt), DwtSTextGetEditable (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSTextClearSelection (3Dwt)

Name

DwtSTextClearSelection – Clears the global selection highlighted in the simple text widget.

Syntax

void DwtSTextClearSelection(*widget*, *time*)
Widget *widget*;
Time *time*;

Arguments

widget	Specifies the widget ID.
time	Specifies the time of the event that led to the call to `XSetSelectionOwner`. You can pass either a timestamp or `CurrentTime`. Whenever possible, however, use the timestamp of the event leading to the call.

Description

The `DwtSTextClearSelection` function clears the global selection highlighted in the simple text widget.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSTextGetEditable – Obtains the current edit permission state indicating whether the user can edit the text in the simple text widget.

Syntax

Boolean DwtSTextGetEditable(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the simple text widget whose edit permission state you want to obtain.

Description

The `DwtSTextGetEditable` function returns the current edit-permission-state, which indicates whether the user can edit the text in the simple text widget. If the function returns `True`, the user can edit the string text in the simple text widget. If it returns `False`, the user cannot edit the text.

Return Value

This function returns the current permission state concerning the editing of the text field in the widget.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSTextGetMaxLength (3Dwt)

Name

DwtSTextGetMaxLength – Gets the current maximum allowable length of the text string in the simple text widget.

Syntax

int DwtSTextGetMaxLength(*widget*)
Widget *widget*;

Arguments

widget	Specifies the ID of the simple text widget whose maximum text string length you want to obtain.

Description

The `DwtSTextGetMaxLength` function returns the current maximum allowable length of the text string in the simple text widget.

Return Value

This function returns the maximum length of the text widget.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSTextGetSelection – Retrieves the global selection, if any, currently highlighted in the simple text widget.

Syntax

```
char *DwtSTextGetSelection(widget)
Widget widget;
```

Arguments

widget	Specifies the widget ID.

Description

The `DwtSTextGetSelection` function retrieves the text currently highlighted (selected) in the simple text widget. It returns a NULL-pointer if no text is selected in the widget. The application is responsible for freeing the storage associated with the string by calling `XtFree`.

Return Value

This function returns the text currently highlighted on the screen.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSTextGetString (3Dwt)

Name

DwtSTextGetString – Retrieves the text string from the simple text widget.

Syntax

```
char *DwtSTextGetString(widget)
    Widget widget;
```

Arguments

widget	Specifies the ID of the simple text widget

Description

The `DwtSTextGetString` function returns a pointer to the current string in the simple text widget window. The application is responsible for freeing the storage associated with the string by calling `XtFree`.

Return Value

This function returns the pointer to the string currently displayed in the given text widget window.

See Also

DwtSTextSetString (3Dwt), DwtSTextReplace (3Dwt), DwtSTextGetEditable (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSTextReplace – Replaces a portion of the current text string in the simple text widget or inserts a new substring in the text.

Syntax

void DwtSTextReplace(*widget*, *from_pos*, *to_pos*, value)
Widget *widget*;
int *from_pos*, *to_pos*;
DwtCompString *value*;

Arguments

widget	Specifies the ID of the simple text widget whose text string you want to replace.
from_pos	Specifies the beginning character position within the text string marking the text being replaced.
to_pos	Specifies the last character position within the text string marking the text being replaced.
value	Specifies the text to replace part of the current text in the simple text widget.

Description

The `DwtSTextReplace` function replaces part of the text string in the simple text widget. Within the window, the positions are numbered starting from 0 and increasing sequentially. For example, to replace the second and third characters in the string, *from_pos* should be 1 and *to_pos* should be 3. To insert a string after the fourth character, *from_pos* and *to_pos* should both be 4.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextGetEditable (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSTextSetEditable (3Dwt)

Name

DwtSTextSetEditable – Sets the permission state that determines whether the user can edit text in the simple text widget.

Syntax

void DwtSTextSetEditable(*widget*, *editable*)
Widget *widget*;
Boolean *editable*;

Arguments

widget	Specifies the ID of the simple text widget whose edit permission state you want to set.
editable	Specifies a boolean value that, when `True`, indicates that the user can edit the text string in the simple text widget. If `False`, prohibits the user from editing the text string.

Description

The `DwtSTextSetEditable` function sets the edit permission state information concerning whether the user can edit text in the simple text widget.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSTextSetMaxLength – Sets the maximum allowable length of the text string in the simple text widget.

Syntax

void DwtSTextSetMaxLength(*widget*, *max_length*)
Widget *widget*;
int *max_length*;

Arguments

widget	Specifies the ID of the simple text widget whose maximum text string length you want to set.
max_length	Specifies the maximum length of the text string in the simple text widget. This argument sets the `DwtNmaxLength` attribute associated with `DwtSTextCreate`.

Description

The `DwtSTextSetMaxLength` function sets the maximum allowable length of the text in the simple text widget and prevents the user from entering text larger than this limit.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtSTextSetSelection (3Dwt)

Name

DwtSTextSetSelection – Makes the specified text in the simple text widget the current global selection and highlights it in the simple text widget.

Syntax

void DwtSTextSetSelection(*widget*, *first*, *last*, *time*)
Widget *widget*;
int *first*, *last*;
Time *time*;

Arguments

widget	Specifies the widget ID.
first	Specifies the first character position of the selected string.
last	Specifies the last character position of the selected string.
time	Specifies the time of the event that led to the call to `XSetSelectionOwner`. You can pass either a timestamp or `CurrentTime`. Whenever possible, however, use the timestamp of the event leading to the call.

Description

The `DwtSTextSetSelection` function makes the specified text in the simple text widget the current global selection and highlights it in the simple text widget. Within the text window, *first* marks the first character position and *last* marks the last position. The field characters are numbered in sequence starting at 0.

See Also

DwtSText (3Dwt), DwtSTextGetString (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextGetSelection (3Dwt), DwtSTextClearSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtSTextSetString – Sets the text string in the simple text widget.

Syntax

void DwtSTextSetString(*widget*, *value*)
Widget *widget*;
char **value*;

Arguments

widget	Specifies the ID of the simple text widget whose text string you want to set.
value	Specifies the text that replaces all text in the current text widget window.

Description

The `DwtSTextSetString` function completely changes the string in the simple text widget.

See Also

DwtSTextGetString (3Dwt), DwtSTextReplace (3Dwt), DwtSTextGetEditable (3Dwt), DwtSTextSetEditable (3Dwt), DwtSTextGetMaxLength (3Dwt), DwtSTextSetMaxLength (3Dwt), DwtSTextSetSelection (3Dwt), DwtSTextClearSelection (3Dwt), DwtSTextGetSelection (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtString (3Dwt)

Name

DwtString – Creates a compound-string.

Syntax

DwtCompString DwtString(*text*, *charset*, *direction_r_to_l*)
char **text*;
unsigned long *charset*;
char *direction_r_to_l*;

Arguments

text Specifies the text string to be converted to a compound-string.

charset Specifies the character set for the compound-string. Values for this argument can be found in the required file `/usr/include/cda_def.h`.

direction_r_to_l
Specifies the direction in which the text is drawn and wraps. You can pass `DwtDirectionLeftDown` (text is drawn from left to right and wraps down); `DwtDirectionRightUp` (text is drawn from left to right and wraps up); `DwtDirectionLeftDown` (text is drawn from right to left and wraps down); or `DwtDirectionLeftUp` (text is drawn from right to left and wraps up).

Description

The `DwtString` function creates a compound-string from information in the argument list. It has a simpler interface than the one used for `DwtCSString`.

`DwtString` assumes the following default values:

- For *language* the default is `DwtLanguageNotSpecified`.
- For *rend* the default is `DwtRendMaskNone`. The space for the resulting compound-string is allocated within the function. After using this function, you should free this space by calling `XtFree`.

Return Value

This function returns the resulting compound-string. However, it returns a NULL pointer if the text is NULL.

See Also

DwtCSString (3Dwt), DwtLatin1String (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtStringFreeContext (3Dwt)

Name

DwtStringFreeContext – Frees a compound-string context structure.

Syntax

void DwtStringFreeContext(*context*)
DwtCompStringContext **context*;

Arguments

context	Specifies the compound-string context structure initialized by `DwtStringInitContext`.

Description

The `DwtStringFreeContext` function frees the compound-string context structure returned by `DwtStringInitContext`. When your application has finished with the context, it should call `DwtStringFreeContext`.

See Also

DwtStringInitContext (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtStringInitContext – Initializes a compound-string context structure needed by `DwtGetNextSegment`

Syntax

Boolean DwtStringInitContext(*context*, *compound_string*)
DwtCompStringContext *context*;
DwtCompString *compound_string*;

Arguments

context Specifies the compound-string context structure initialized by `DwtStringInitContext`.

compound_string
Specifies the compound-string.

Description

The `DwtStringInitContext` function initializes a compound-string context structure. The context structure is needed for calling `DwtGetNextSegment`. For performance reasons, `DwtStringInitContext` is preferred over `DwtInitGetSegment`.

After fetching the necessary segments using `DwtGetNextSegment`, call `DwtStringFreeContext` to free the context structure.

Return Value

This function returns one of these status return constants:

`True`	The compound-string context structure has been successfully initialized.
`False`	The compound-string context structure has not been successfully initialized.

See Also

DwtGetNextSegment (3Dwt), DwtStringFreeContext (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtToggleButton(3Dwt)

Name

DwtToggleButton, DwtToggleButtonCreate – Creates a toggle button widget for the application to display screen settable switches for the user.

Syntax

Widget DwtToggleButton(*parent_widget*, *name*, *x*, *y*,
label, *value*, *callback*, *help_callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
DwtCompString *label*;
Boolean *value*;
DwtCallbackPtr *callback*;
DwtCallbackPtr *help_callback*;

Widget DwtToggleButtonCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
label	Specifies the text in the toggle button label/indicator. This argument sets the `DwtNlabel` attribute associated with `DwtLabelCreate`.
value	Specifies a boolean value that, when `False`, indicates the button state is off. If `True`, the button state is on. This

argument sets the DwtNvalue attribute associated with DwtToggleButtonCreate.

callback Specifies the callback function or functions called back when the value of the toggle button changes. This argument sets the DwtNarmCallback, DwtNdisarmCallback, and DwtNvalueChangedCallback attributes associated with DwtToggleButtonCreate.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the DwtNhelpCallback common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtToggleButton and DwtToggleButtonCreate functions create an instance of a toggle button widget and return its associated widget ID. When calling DwtToggleButton, you set the common toggle button widget attributes presented in the formal parameter list. For DwtToggleButtonCreate, however, you specify a list of attribute name/value pairs that represent all the possible toggle button widget attributes.

The toggle button widget consists of either a label and indicator button combination or simply a pixmap (icon). Toggle buttons imply an on or off state. These functions use their attributes to configure the visual representation, "looks," and the user interface syntax "feel," for the application. Note that the callback data structure includes a value member, which allows the callback data function to pass the status of the toggle switch back to the application.

The sizing is affected by spacing, font (affects indicator), and label. See the description for DwtLabel and DwtLabelCreate.

The sizing is affected by these attributes: DwtNspacing, DwtNfont (text label), and DwtNlabel. For more information, see DwtLabel and DwtLabelCreate.

The DwtNindicator size is based on the height of the toggle button minus twice the margin height. The DwtNindicator width is equal to the indicator height.

The default margin height is four pixels. The default margin width is five pixels.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Width of the label or pixmap, plus three times DwtNmarginWidth, plus the width of DwtNindicator
DwtNheight	Dimension	The height of the label or pixmap, plus two times DwtNmarginHeight
DwtNborderWidth	Dimension	zero pixels
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Label Attributes

DwtNlabelType	unsigned char	DwtCString
DwtNlabel	DwtCompString	Widget name
DwtNmarginWidth	Dimension	Two pixels for text Zero pixels for pixmap
DwtNmarginHeight	Dimension	Two pixels for text Zero pixels for pixmap
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNpixmap	Pixmap	NULL
DwtNmarginLeft	Dimension	Zero
DwtNmarginRight	Dimension	Zero
DwtNmarginTop	Dimension	Zero
DwtNmarginBottom	Dimension	Zero
DwtNconformToText	Boolean	True, if the widget is created with a width and height of zero False, if the widget is created with a non-zero width and height

Widget-Specific Attributes

You can set the following widget-specifc attributes in the *override_arglist*:

Attribute Name	Data Type	Default
DwtNshape	unsigned char	DwtRectangular
DwtNvisibleWhenOff	Boolean	True
DwtNspacing	short	4 pixels
DwtNpixmapOn	Pixmap	NULL
DwtNpixmapOff	Pixmap	NULL
DwtNvalue	Boolean	False

DwtNarmCallback	DwtCallbackPtr	NULL
DwtNdisarmCallback	DwtCallbackPtr	NULL
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNindicator	Boolean	True when the label is DwtCString False when the label is DwtPixmap
DwtNacceleratorText	DwtCompString	NULL
DwtNbuttonAccelerator	char *	NULL
DwtNinsensitivePixmapOn	Pixmap	NULL
DwtNinsensitivePixmapOff	Pixmap	NULL

DwtNshape
: Specifies the toggle button indicator shape. You can pass DwtRectangular or DwtOval.

DwtNvisibleWhenOff
: Specifies a boolean value that, when True, indicates that the toggle button is visible when in the off state.

DwtNspacing
: Specifies the number of pixels between the label and the button if DwtNlabelType is DwtCompString.

DwtNpixmapOn
: Specifies the pixmap to be used as the button label if DwtNlabelType is DwtPixmap and the toggle button is in the on state.

DwtNpixmapOff
: Specifies the pixmap to be used as the button label if DwtNlabelType is DwtPixmap and the toggle button is in the off state.

DwtNvalue
: Specifies a boolean value that, when False, indicates the button state is off. If True, the button state is on.

DwtNarmCallback
: Specifies the callback function or functions called when the toggle button is armed. The toggle button is armed when the user presses and releases MB1 while the pointer is inside the toggle button widget. For this callback, the reason is DwtCRArm.

DwtNdisarmCallback
: Specifies the callback function or functions called when the button is disarmed. The button is disarmed when the user presses MB1 while the pointer is inside the toggle button widget, but moves the

pointer outside the toggle button before releasing MB1. For this callback, the reason is DwtCRDisarm.

DwtNvalueChangedCallback
: Specifies the callback function or functions called when the toggle button value was changed. For this callback, the reason is DwtCRValueChanged.

DwtNindicator
: Specifies a boolean value that, when True, signifies that the indicator is present in the toggle button. If False, signifies that the indicator is not present in the toggle button.

DwtNacceleratorText
: Specifies the compound-string text displayed for the accelerator.

DwtNbuttonAccelerator
: Sets an accelerator on a toggle button widget.

DwtNinsensitivePixmapOn
: Specifies the pixmap used when the toggle button is on and is insensitive. This attribute applies only if the toggle button label is specified as a pixmap.

DwtNinsensitivePixmapOff
: Specifies the pixmap used when the toggle button is off and is insensitive. This attribute applies only if the toggle button label is specified as a pixmap.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        int value;
} DwtTogglebuttonCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRValueChanged`	The user activated the toggle button to change state.
`DwtCRArm`	The user armed the toggle button by pressing MB1 while the pointer was inside the toggle button widget.
`DwtCRDisarm`	The user disarmed the toggle button by pressing MB1 while the pointer was inside the toggle button widget, but did not release it until after moving the pointer outside the toggle button widget.
`DwtCRHelpRequested`	The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

The value member is set to the toggle button's current state when the callback occurred, either `True` (on) or `False` (off).

See Also

DwtToggleButtonGetState (3Dwt), DwtToggleButtonSetState (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtToggleButtonGadgetCreate – Creates a toggle button gadget.

Syntax

Widget DwtToggleButtonGadgetCreate (*parent_widget*, *name*,
override_arglist,
override_argcount)

Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtToggleButtonGadgetCreate` function creates an instance of the toggle button gadget and returns its associated gadget ID. A toggle button gadget is similar in appearance and semantics to a toggle button widget. Like all gadgets, `DwtToggleButtonGadgetCreate` does not have a window but uses the window of the closest antecedent widget. Thus, the antecedent widget provides all event dispatching for the gadget. This currently restricts gadgets to being descendents of menu or dialog class (or subclass) widgets.

Inherited Attributes

Attribute Name	Data Type	Default
Rectangle Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	The width of the label plus margins
DwtNheight	Dimension	The height of the label plus margins
DwtNborderWidth	Dimension	zero
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
Label Attributes		
DwtNlabel	DwtCompString	Widget name
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNdirectionRToL	Boolean	False
DwtNhelpCallback	DwtCallbackPtr	NULL

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNshape	unsigned char	DwtRectangular
DwtNvalue	Boolean	False
DwtNvisibleWhenOff	Boolean	True
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNbuttonAccelerator	char *	NULL
DwtNacceleratorText	DwtCompString	NULL

DwtNshape — Specifies the toggle button indicator shape. You can pass DwtRectangular or DwtOval.

DwtNvalue — Specifies a boolean value that, when False,

indicates the button state is off. If `True`, the button state is on.

`DwtNvisibleWhenOff`
: Specifies a boolean value that, when `True`, indicates that the toggle button is visible when in the off state.

`DwtNvalueChangedCallback`
: Specifies the callback function or functions called when the toggle button value was changed. For this callback, the reason is `DwtCRValueChanged`.

`DwtNbuttonAccelerator`
: Sets an accelerator on a toggle button widget. This is the same as the `DwtNtranslations` core attribute except that only the left side of the table is to be passed as a character string, not compiled. The application is responsible for calling `XtInstallAllAccelerators` to install the accelerator where the application needs it.

`DwtNacceleratorText`
: Specifies the compound-string text displayed for the accelerator.

Return Value

This function returns the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                        int reason;
                        XEvent *event;
                        int value;
} DwtTogglebuttonCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRValueChanged`	The user activated the toggle button to change state.
`DwtCRHelpRequested`	The user selected Help.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

The value member is set to the toggle button's current state when the callback occurred, either `True` (on) or `False` (off).

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtToggleButtonGetState – Gets the current state of the toggle button.

Syntax

Boolean DwtToggleButtonGetState(*widget*)
Widget *widget*;

Arguments

widget	Specifies the widget ID.

Description

The `DwtToggleButtonGetState` function returns the current state (value) of the toggle button, either `True` (on) or `False` (off).

Return Value

This function returns the toggle button's current state: `True` (on) or `False` (off).

See Also

DwtToggleButton (3Dwt), DwtToggleButtonSetState (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

DwtToggleButtonSetState (3Dwt)

Name

DwtToggleButtonSetState – Sets or changes the current state of the toggle button.

Syntax

void DwtToggleButtonSetState(*widget*, *value*, *notify*)
Widget *widget*;
Boolean *value*;
Boolean *notify*;

Arguments

widget	Specifies the widget ID.
value	Specifies a boolean value that, when `False`, indicates the button state is off. If `True`, the button state is on. This argument sets the `DwtNvalue` attribute associated with `DwtToggleButtonCreate`.
notify	Specifies a boolean value that, when `True`, indicates a recent change in the on/off state of the toggle button and `DwtNvalueChangedCallback` should be activated with the recent change. If `False`, no change in state has occurred and `DwtNvalueChangedCallback` should not be activated.

Description

The `DwtToggleButtonSetState` function sets or changes the toggle button's current state (value) within the display.

See Also

DwtToggleButton (3Dwt), DwtToggleButtonGetState (3Dwt)
Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtUndoCopyToClipboard – Deletes the last item placed on the clipboard.

Syntax

int DwtUndoCopyToClipboard(*display*, *window*)
Display **display*;
Window *window*;

Arguments

display	Specifies a pointer to the `Display` structure that was returned in a previous call to `XOpenDisplay`. For information on `XOpenDisplay` and the `Display` structure, see the *Guide to the Xlib Library: C Language Binding*.
window	Specifies the window ID that relates the application window to the clipboard. The same application instance should pass the same window ID to each clipboard function that it calls.

Description

The `DwtUndoCopyToClipboard` function deletes the last item placed on the clipboard if the item was placed there by an application with the passed *display* and *window* arguments. Any data item deleted from the clipboard by the original call to `DwtCopyToClipboard` is restored. If the *display* or *window* IDs do not match the last copied item, no action is taken and this function has no effect.

Return Value

This function returns one of these status return constants:

`ClipboardSuccess`	The function is successful.
`ClipboardLocked`	The function failed because the clipboard was locked by another application. The application can continue to call the function with the same parameters until the clipboard is unlocked. Optionally, the application can ask if the user wants to keep trying or to give up on the operation.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtWindow, DwtWindowCreate – Creates a window widget for simple applications to display in the main window widget work area.

Syntax

Widget DwtWindow(*parent_widget*, *name*, *x*, *y*, *width*, *height*, *callback*)
Widget *parent_widget*;
char **name*;
Position *x*, *y*;
Dimension *width*, *height*;
DwtCallbackPtr *callback*;

Widget DwtWindowCreate (*parent_widget*, *name*, *override_arglist*, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget	Specifies the parent widget ID.
name	Specifies the name of the created widget.
x	Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.
y	Specifies, in pixels, the placement of the top of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.
width	Specifies in pixels the width of the widget window. This argument sets the `DwtNwidth` core widget attribute.
height	Specifies in pixels the height of the widget window. This argument sets the `DwtNheight` core widget attribute.
callback	Specifies the callback function or functions called when an `Expose` event occurs. This argument sets the `DwtNexposeCallback` attribute associated with

DwtWindowCreate.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The DwtWindow and DwtWindowCreate functions create an instance of the window widget and return its associated widget ID. The window widget simplifies programming allowing you to create an application display directly in the main window widget work area. the window widget simplifies programming by allowing you to create an application display directly in the main window widget work area. When calling DwtWindow, you set the window widget attributes presented in the formal parameter list. For DwtWindowCreate, you specify a list of attribute name/value pairs that represent all the possible window widget attributes.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Determined by the geometry manager
DwtNy	Position	Determined by the geometry manager
DwtNwidth	Dimension	Widget-specific
DwtNheight	Dimension	Widget-specific
DwtNborderWidth	Dimension	One pixel
DwtNborder	Pixel	Default foreground color
DwtNborderPixmap	Pixmap	NULL
DwtNbackground	Pixel	Default background color
DwtNbackgroundPixmap	Pixmap	NULL
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True Setting the sensitivity of the window causes all widgets contained in that window to be set to the same sensitivity as the window.

DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

Common Attributes

DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL
DwtNuserData	Opaque *	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	NOT SUPPORTED	

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNexposeCallback	DwtCallbackPtr	NULL

DwtNexposeCallback

Specifies the callback function or functions called when the window had an Expose event. For this callback, the reason is DwtCRExpose.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XExposeEvent *event;
                    Window w;
} DwtWindowCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRExpose`	The window widget had an `Expose` event.

The event member is a pointer to the Xlib structure `XExposeEvent`. This structure is associated with exposure event processing, and, specifically, with `Expose` events. For information on exposure event processing, see the *Guide to the Xlib Library: C Language Binding*.

The members of the `XExposeEvent` structure associated with `Expose` events are window, x, y, width, height, and count. The window member is set to the window ID of the exposed (damaged) window. The x and y members are set to the coordinates relative to the drawable's origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of `Expose` events that are to follow. If count is set to zero (0), no more `Expose` events follow for this window. However, if count is set to nonzero, at least count `Expose` events and possibly more follow for this window. Simple applications that do not want to optimize redisplay by distinguishing between subareas of its windows can just ignore all `Expose` events with nonzero counts and perform full redisplays on events with zero counts.

The w member is set to the X window ID where the `Expose` event occurred.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Name

DwtWorkBox, DwtWorkBoxCreate – Creates a work-in-progress box widget for the application to display work in progress messages.

Syntax

Widget DwtWorkBox(*parent_widget*, *name*, *default_position*,
x, *y*, *style*, *label*, *cancel_label*,
callback, *help_callback*)
Widget *parent_widget*;
char **name*;
Boolean *default_position*;
Position *x*, *y*;
int *style*;
DwtCompString *label*, *cancel_label*;
DwtCallbackPtr *callback*, *help_callback*;

Widget DwtWorkBoxCreate (*parent_widget*, *name*,
override_arglist, *override_argcount*)
Widget *parent_widget*;
char **name*;
ArgList *override_arglist*;
int *override_argcount*;

Arguments

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

default_position
Specifies a boolean value that, when `True`, causes `DwtNx` and `DwtNy` to be ignored and forces the default widget position. The default widget position is centered in the parent window. If `False`, the specified `DwtNx` and `DwtNy` attributes are used to position the widget. This argument sets the `DwtNdefaultPosition` attribute associated with `DwtDialogBoxCreate`.

x Specifies the placement, in pixels, of the left side of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNx` core widget attribute.

y Specifies, in pixels, the placement of the upper left corner of the widget window relative to the inner upper left corner of the parent window. This argument sets the `DwtNy` core widget attribute.

style Specifies the style of the dialog box widget. You can pass `DwtModal` (modal) or `DwtModeless` (modeless). This argument sets the `DwtNstyle` attribute associated with `DwtDialogBoxPopupCreate`.

label Specifies the text in the message line or lines. This argument sets the `DwtNlabel` attribute associated with `DwtWorkBoxCreate`.

cancel_label Specifies the label for the Cancel push button. If the label is a NULL string, the button is not displayed. This argument sets the `DwtNcancelLabel` attribute associated with `DwtWorkBoxCreate`.

callback Specifies the callback function or functions called back when the Cancel button is activated. This argument sets the `DwtNcancelCallback` attribute associated with `DwtWorkBoxCreate`.

help_callback Specifies the callback function or functions called when a help request is made. This argument sets the `DwtNhelpCallback` common widget attribute.

override_arglist
Specifies the application override argument list.

override_argcount
Specifies the number of attributes in the application override argument list (*override_arglist*).

Description

The `DwtWorkBox` and `DwtWorkBoxCreate` functions create an instance of a work-in-progress box widget and return its associated widget ID. When calling `DwtWorkBox`, you set the work-in-progress box widget attributes presented in the formal parameter list. For `DwtWorkBoxCreate`, however, you specify a list of attribute name/value pairs that represent all the possible work-in-progress box widget attributes. The work-in-progress box widget is a dialog box that allows the application to display work in progress messages to the user. When the application determines that an operation will take longer than five seconds, it is recommended that the application call this function to display a work-in-progress box with a message such as **"Work in**

Progress./Please Wait." The work-in-progress box may contain a push button labeled "Cancel Operation." Do not include the push button if the operation cannot be canceled. If the style is `DwtModal` when the user selects the Cancel push button, the widget is cleared from the screen, but not destroyed. The widget can be redisplayed by calling `XtManageChild`.

Inherited Attributes

Attribute Name	Data Type	Default
Core Attributes		
`DwtNx`	`Position`	Determined by the geometry manager
`DwtNy`	`Position`	Determined by the geometry manager
`DwtNwidth`	`Dimension`	5 pixels
`DwtNheight`	`Dimension`	5 pixels
`DwtNborderWidth`	`Dimension`	One pixel
`DwtNborder`	`Pixel`	Default foreground color
`DwtNborderPixmap`	`Pixmap`	NULL
`DwtNbackground`	`Pixel`	Default background color
`DwtNbackgroundPixmap`	`Pixmap`	NULL
`DwtNcolormap`	`Colormap`	Default color map
`DwtNsensitive`	`Boolean`	`True`
`DwtNancestorSensitive`	`Boolean`	The bitwise AND of the parent widget's `DwtNsensitive` and `DwtNancestorSensitive` attributes
`DwtNaccelerators`	`XtTranslations`	NULL
`DwtNdepth`	int	Depth of the parent window
`DwtNtranslations`	`XtTranslations`	NULL
`DwtNmappedWhenManaged`	`Boolean`	`True`
`DwtNscreen`	`Screen *`	The parent screen
`DwtNdestroyCallback`	`DwtCallbackPtr`	NULL
Dialog Pop-Up Attributes		
`DwtNforeground`	`Pixel`	Default foreground color
`DwtNhighlight`	`Pixel`	Default foreground color
`DwtNhighlightPixmap`	`Pixmap`	NULL
`DwtNuserData`	`Opaque *`	NULL

DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	NOT SUPPORTED	
DwtNunits	NOT SUPPORTED	
DwtNtitle	DwtCompString	Widget name
DwtNstyle	unsigned char	DwtModal
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	NOT SUPPORTED	
DwtNmarginWidth	Dimension	12 pixels
DwtNmarginHeight	Dimension	10 pixels
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	NOT SUPPORTED	
DwtNresize	unsigned char	DwtResizeShrinkWrap
DwtNtakeFocus	Boolean	True for modal dialog box False for modeless dialog box
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	NOT SUPPORTED	
DwtNcancelButton	NOT SUPPORTED	

Widget-Specific Attributes

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNcancelLabel	DwtCompString	"Cancel"
DwtNcancelCallback	DwtCallbackPtr	NULL

DwtNlabel — Specifies the text in the message line or lines.

DwtNcancelLabel — Specifies the label for the Cancel push button. If the label is a NULL string, the button is not displayed.

DwtNcancelCallback — Specifies the callback function or functions called when the user clicks on the Cancel button. For this callback, the reason is DwtCRCancel.

Return Value

These functions return the ID of the created widget.

Callback Information

The following structure is returned to your callback:

```
typedef struct {
                    int reason;
                    XEvent *event;
} DwtAnyCallbackStruct;
```

The reason member is set to a constant that represents the reason why this callback was invoked. For this callback, the reason member can be set to:

`DwtCRCancel`	The user activated the cancel push button.
`DwtCRFocus`	The work-in-progress box has received the input focus.
`DwtCRHelpRequested`	The user selected Help somewhere in the work-in-progress box.

The event member is a pointer to the Xlib structure `XEvent`, which describes the event that generated this callback. This structure is a union of the individual structures declared for each event type. For information on `XEvent` and event processing, see the *Guide to the Xlib Library: C Language Binding*.

See Also

Guide to the XUI Toolkit: C Language Binding
Guide to the XUI Toolkit Intrinsics: C Language Binding

Insert tabbed
divider here.
Then discard
this sheet.

Name

AllPlanes, BlackPixel, WhitePixel, ConnectionNumber, DefaultColormap, DefaultDepth, DefaultGC, DefaultRootWindow, DefaultScreenOfDisplay, DefaultScreen, DefaultVisual, DisplayCells, DisplayPlanes, DisplayString, LastKnownRequestProcessed, NextRequest, ProtocolVersion, ProtocolRevision, QLength, RootWindow, ScreenCount, ScreenOfDisplay, ServerVendor, VendorRelease – Display macros

Syntax

AllPlanes()

BlackPixel(*display*, *screen_number*)

WhitePixel(*display*, *screen_number*)

ConnectionNumber(*display*)

DefaultColormap(*display*, *screen_number*)

DefaultDepth(*display*, *screen_number*)

DefaultGC(*display*, *screen_number*)

DefaultRootWindow(*display*)

DefaultScreenOfDisplay(*display*)

DefaultScreen(*display*)

DefaultVisual(*display*, *screen_number*)

DisplayCells(*display*, *screen_number*)

DisplayPlanes(*display*, *screen_number*)

DisplayString(*display*)

LastKnownRequestProcessed(*display*)

NextRequest(*display*)

ProtocolVersion(*display*)

ProtocolRevision(*display*)

QLength(*display*)

RootWindow(*display*, *screen_number*)

ScreenCount(*display*)

AllPlanes (3X11)

ScreenOfDisplay(*display*, *screen_number*)

ServerVendor(*display*)

VendorRelease(*display*)

Arguments

display Specifies the connection to the X server.

screen_number Specifies the appropriate screen number on the host server.

Description

The `AllPlanes` macro returns a value with all bits set to 1 suitable for use in a plane argument to a procedure.

The `BlackPixel` macro returns the black pixel value for the specified screen.

The `WhitePixel` macro returns the white pixel value for the specified screen.

The `ConnectionNumber` macro returns a connection number for the specified display.

The `DefaultColormap` macro returns the default colormap ID for allocation on the specified screen.

The `DefaultDepth` macro returns the depth (number of planes) of the default root window for the specified screen.

The `DefaultGC` macro returns the default GC for the root window of the specified screen.

The `DefaultRootWindow` macro returns the root window for the default screen.

The `DefaultScreenOfDisplay` macro returns the default screen of the specified display.

The `DefaultScreen` macro returns the default screen number referenced in the `XOpenDisplay` routine.

The `DefaultVisual` macro returns the default visual type for the specified screen.

The `DisplayCells` macro returns the number of entries in the default colormap.

The `DisplayPlanes` macro returns the depth of the root window of the specified screen.

The `DisplayString` macro returns the string that was passed to `XOpenDisplay` when the current display was opened.

The `LastKnownRequestProcessed` macro extracts the full serial number of the last request known by Xlib to have been processed by the X server.

The `NextRequest` macro extracts the full serial number that is to be used for the next request.

The `ProtocolVersion` macro returns the major version number (11) of the X protocol associated with the connected display.

The `ProtocolRevision` macro returns the minor protocol revision number of the X server.

The `QLength` macro returns the length of the event queue for the connected display.

The `RootWindow` macro returns the root window.

The `ScreenCount` macro returns the number of available screens.

The `ScreenOfDisplay` macro returns a pointer to the screen of the specified display.

The `ServerVendor` macro returns a pointer to a null-terminated string that provides some identification of the owner of the X server implementation.

The `VendorRelease` macro returns a number related to a vendor's release of the X server.

See Also

BlackPixelOfScreen(3X11), ImageByteOrder(3X11), IsCursorKey(3X11)
Guide to the Xlib Library

BlackPixelOfScreen (3X11)

Name

BlackPixelOfScreen, WhitePixelOfScreen, CellsOfScreen, DefaultColormapOfScreen, DefaultDepthOfScreen, DefaultGCOfScreen, DefaultVisualOfScreen, DoesBackingStore, DoesSaveUnders, DisplayOfScreen, EventMaskOfScreen, HeightOfScreen, HeightMMOfScreen, MaxCmapsOfScreen, MinCmapsOfScreen, PlanesOfScreen, RootWindowOfScreen, WidthOfScreen, WidthMMOfScreen – screen information macros

Syntax

BlackPixelOfScreen(*screen*)

WhitePixelOfScreen(*screen*)

CellsOfScreen(*screen*)

DefaultColormapOfScreen(*screen*)

DefaultDepthOfScreen(*screen*)

DefaultGCOfScreen(*screen*)

DefaultVisualOfScreen(*screen*)

DoesBackingStore(*screen*)

DoesSaveUnders(*screen*)

DisplayOfScreen(*screen*)

EventMaskOfScreen(*screen*)

HeightOfScreen(*screen*)

HeightMMOfScreen(*screen*)

MaxCmapsOfScreen(*screen*)

MinCmapsOfScreen(*screen*)

PlanesOfScreen(*screen*)

RootWindowOfScreen(*screen*)

WidthOfScreen(*screen*)

WidthMMOfScreen(*screen*)

Arguments

screen Specifies a pointer to the appropriate `Screen` structure.

Description

The `BlackPixelOfScreen` macro returns the black pixel value of the specified screen.

The `WhitePixelOfScreen` macro returns the white pixel value of the specified screen.

The `CellsOfScreen` macro returns the number of colormap cells in the default colormap of the specified screen.

The `DefaultColormapOfScreen` macro returns the default colormap of the specified screen.

The `DefaultDepthOfScreen` macro returns the default depth of the root window of the specified screen.

The `DefaultGCOfScreen` macro returns the default GC of the specified screen, which has the same depth as the root window of the screen.

The `DefaultVisualOfScreen` macro returns the default visual of the specified screen.

The `DoesBackingStore` macro returns `WhenMapped`, `NotUseful`, or `Always`, which indicate whether the screen supports backing stores.

The `DoesSaveUnders` macro returns a Boolean value indicating whether the screen supports save unders.

The `DisplayOfScreen` macro returns the display of the specified screen.

The `EventMaskOfScreen` macro returns the root event mask of the root window for the specified screen at connection setup time.

The `HeightOfScreen` macro returns the height of the specified screen.

The `HeightMMOfScreen` macro returns the height of the specified screen in millimeters.

The `MaxCmapsOfScreen` macro returns the maximum number of installed colormaps supported by the specified screen.

The `MinCmapsOfScreen` macro returns the minimum number of installed colormaps supported by the specified screen.

The `PlanesOfScreen` macro returns the number of planes in the root window of the specified screen.

BlackPixelOfScreen (3X11)

The `RootWindowOfScreen` macro returns the root window of the specified screen.

The `WidthOfScreen` macro returns the width of the specified screen.

The `WidthMMOfScreen` macro returns the width of the specified screen in millimeters.

See Also

AllPlanes(3X11), ImageByteOrder(3X11), IsCursorKey(3X11)
Guide to the Xlib Library

Name

ImageByteOrder, BitmapBitOrder, BitmapPad, BitmapUnit, DisplayHeight, DisplayHeightMM, DisplayWidth, DisplayWidthMM – image format macros

Syntax

ImageByteOrder(*display*)

BitmapBitOrder(*display*)

BitmapPad(*display*)

BitmapUnit(*display*)

DisplayHeight(*display*, *screen_number*)

DisplayHeightMM(*display*, *screen_number*)

DisplayWidth(*display*, *screen_number*)

DisplayWidthMM(*display*, *screen_number*)

Arguments

display Specifies the connection to the X server.

screen_number Specifies the appropriate screen number on the host server.

Description

The `ImageByteOrder` macro specifies the required byte order for images for each scanline unit in XY format (bitmap) or for each pixel value in Z format.

The `BitmapBitOrder` macro returns `LSBFirst` or `MSBFirst` to indicate whether the leftmost bit in the bitmap as displayed on the screen is the least or most significant bit in the unit.

The `BitmapPad` macro returns the number of bits that each scanline must be padded.

The `BitmapUnit` macro returns the size of a bitmap's scanline unit in bits.

The `DisplayHeight` macro returns the height of the specified screen in pixels.

The `DisplayHeightMM` macro returns the height of the specified screen in millimeters.

The `DisplayWidth` macro returns the width of the screen in pixels.

The `DisplayWidthMM` macro returns the width of the specified screen in millimeters.

See Also

AllPlanes(3X11), BlackPixelOfScreen(3X11), IsCursorKey(3X11)
Guide to the Xlib Library

Name

IsCursorKey, IsFunctionKey, IsKeypadKey, IsMiscFunctionKey, IsModiferKey, IsPFKey – keysym classification macros

Syntax

IsCursorKey(*keysym*)

IsFunctionKey(*keysym*)

IsKeypadKey(*keysym*)

IsMiscFunctionKey(*keysym*)

IsModifierKey(*keysym*)

IsPFKey(*keysym*)

Arguments

keysym	Specifies the KeySym that is to be tested.

Description

The `IsCursorKey` macro returns `True` if the specified KeySym is a cursor key.

The `IsFunctionKey` macro returns `True` if the KeySym is a function key.

The `IsKeypadKey` macro returns `True` if the specified KeySym is a keypad key.

The `IsMiscFunctionKey` macro returns `True` if the specified KeySym is a miscellaneous function key.

The `IsModiferKey` macro returns `True` if the specified KeySym is a modifier key.

The `IsPFKey` macro returns `True` if the specified KeySym is a PF key.

See Also

AllPlanes(3X11), BlackPixelOfScreen(3X11), ImageByteOrder(3X11)
Guide to the Xlib Library

XAddHost (3X11)

Name

XAddHost, XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XSetAccessControl, XEnableAccessControl, XDisableAccessContro – control host access

Syntax

XAddHost(*display*, *host*)
Display **display*;
XHostAddress **host*;

XAddHosts(*display*, *hosts*, *num_hosts*)
Display **display*;
XHostAddress **hosts*;
int *num_hosts*;

XHostAddress *XListHosts(*display*, *nhosts_return*, *state_return*)
Display **display*;
int **nhosts_return*;
Bool **state_return*;

XRemoveHost(*display*, *host*)
Display **display*;
XHostAddress **host*;

XRemoveHosts(*display*, *hosts*, *num_hosts*)
Display **display*;
XHostAddress **hosts*;
int *num_hosts*;

XSetAccessControl(*display*, *mode*)
Display **display*;
int *mode*;

XEnableAccessControl(*display*)
Display **display*;

XDisableAccessControl(*display*)
Display **display*;

Arguments

display	Specifies the connection to the X server.
host	Specifies the host that is to be added or removed.
hosts	Specifies each host that is to be added or removed.

mode	Specifies the mode. You can pass `EnableAccess` or `DisableAccess`.
nhosts_return	Returns the number of hosts currently in the access control list.
num_hosts	Specifies the number of hosts.
state_return	Returns the state of the access control.

Description

The `XAddHost` function adds the specified host to the access control list for that display. The server must be on the same host as the client issuing the command, or a `BadAccess` error results.

`XAddHost` can generate `BadAccess` and `BadValue` errors.

The `XAddHosts` function adds each specified host to the access control list for that display. The server must be on the same host as the client issuing the command, or a `BadAccess` error results.

`XAddHosts` can generate `BadAccess` and `BadValue` errors.

The `XListHosts` function returns the current access control list as well as whether the use of the list at connection setup was enabled or disabled. `XListHosts` allows a program to find out what machines can make connections. It also returns a pointer to a list of host structures that were allocated by the function. When no longer needed, this memory should be freed by calling `XFree`.

The `XRemoveHost` function removes the specified host from the access control list for that display. The server must be on the same host as the client process, or a `BadAccess` error results. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed unless you reset the server.

`XRemoveHost` can generate `BadAccess` and `BadValue` errors.

The `XRemoveHosts` function removes each specified host from the access control list for that display. The X server must be on the same host as the client process, or a `BadAccess` error results. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed unless you reset the server.

`XRemoveHosts` can generate `BadAccess` and `BadValue` errors.

The `XSetAccessControl` function either enables or disables the use of the access control list at each connection setup.

`XSetAccessControl` can generate `BadAccess` and `BadValue` errors.

The `XEnableAccessControl` function enables the use of the access control list at each connection setup.

`XEnableAccessControl` can generate a `BadAccess` error.

The `XDisableAccessControl` function disables the use of the access control list at each connection setup.

`XDisableAccessControl` can generate a `BadAccess` error.

Diagnostics

`BadAccess`	A client attempted to modify the access control list from other than the local (or otherwise authorized) host.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

Name

XAllocColor, XAllocNamedColor, XAllocColorCells, XAllocColorPlanes, XFreeColors – allocate and free colors

Syntax

Status XAllocColor(*display*, *colormap*, *screen_in_out*)
Display **display*;
Colormap *colormap*;
XColor **screen_in_out*;

Status XAllocNamedColor(*display*, *colormap*, *color_name*, *screen_def_return*, *exact_def_return*)
Display **display*;
Colormap *colormap*;
char **color_name*;
XColor **screen_def_return*, **exact_def_return*;

Status XAllocColorCells(*display*, *colormap*, *contig*, *plane_masks_return*, *nplanes*, *pixels_return*, *npixels*)
Display **display*;
Colormap *colormap*;
Bool *contig*;
unsigned long *plane_masks_return*[];
unsigned int *nplanes*;
unsigned long *pixels_return*[];
unsigned int *npixels*;

Status XAllocColorPlanes(*display*, *colormap*, *contig*, *pixels_return*, *ncolors*, *nreds*, *ngreens*, *nblues*, *rmask_return*, *gmask_return*, *bmask_return*)
Display **display*;
Colormap *colormap*;
Bool *contig*;
unsigned long *pixels_return*[];
int *ncolors*;
int *nreds*, *ngreens*, *nblues*;
unsigned long **rmask_return*, **gmask_return*, **bmask_return*;

XFreeColors(*display*, *colormap*, *pixels*, *npixels*, *planes*)
Display **display*;
Colormap *colormap*;
unsigned long *pixels*[];
int *npixels*;
unsigned long *planes*;

Arguments

color_name Specifies the color name string (for example, red) whose color definition structure you want returned.

colormap Specifies the colormap.

contig Specifies a Boolean value that indicates whether the planes must be contiguous.

display Specifies the connection to the X server.

exact_def_return
Returns the exact RGB values.

ncolors Specifies the number of pixel values that are to be returned in the pixels_return array.

npixels Specifies the number of pixels.

nplanes Specifies the number of plane masks that are to be returned in the plane masks array.

nreds
ngreens
nblues
Specify the number of red, green, and blue planes. The value you pass must be nonnegative.

pixels Specifies an array of pixel values.

pixels_return Returns an array of pixel values.

plane_mask_return
Returns an array of plane masks.

planes Specifies the planes you want to free.

rmask_return
gmask_return
bmask_return Return bit masks for the red, green, and blue planes.

screen_def_return
Returns the closest RGB values provided by the hardware.

screen_in_out Specifies and returns the values actually used in the colormap.

Description

The XAllocColor function allocates a read-only colormap entry corresponding to the closest RGB values supported by the hardware. XAllocColor returns the pixel value of the color closest to the specified RGB elements supported by the hardware and returns the RGB values actually used. The corresponding colormap cell is read-only. In addition, XAllocColor returns nonzero if it succeeded or zero if it failed. Read-only colormap cells are shared among clients. When the last client deallocates a shared cell, it is deallocated. XAllocColor does not use or affect the flags in the XColor structure.

XAllocColor can generate a BadColor error.

The XAllocNamedColor function looks up the named color with respect to the screen that is associated with the specified colormap. It returns both the exact database definition and the closest color supported by the screen. The allocated color cell is read-only. You should use the ISO Latin-1 encoding; uppercase and lowercase do not matter.

XAllocNamedColor can generate a BadColor error.

The XAllocColorCells function allocates read/write color cells. The number of colors must be positive and the number of planes nonnegative, or a BadValue error results. If ncolors and nplanes are requested, then ncolors pixels and nplane plane masks are returned. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. By ORing together each pixel with zero or more masks, ncolors * $2^{nplanes}$ distinct pixels can be produced. All of these are allocated writable by the request. For GrayScale or PseudoColor, each mask has exactly one bit set to 1. For DirectColor, each has exactly three bits set to 1. If contig is True and if all masks are ORed together, a single contiguous set of bits set to 1 will be formed for GrayScale or PseudoColor and three contiguous sets of bits set to 1 (one within each pixel subfield) for DirectColor. The RGB values of the allocated entries are undefined. XAllocColorCells returns nonzero if it succeeded or zero if it failed.

XAllocColorCells can generate BadColor and BadValue errors.

The specified ncolors must be positive; and nreds, ngreens, and nblues must be nonnegative, or a BadValue error results. If ncolors colors, nreds reds, ngreens greens, and nblues blues are requested, ncolors pixels are returned; and the masks have nreds, ngreens, and nblues bits set to 1, respectively. If contig is True, each mask will have a contiguous set of bits set to 1. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. For DirectColor, each mask will lie within the

corresponding pixel subfield.
By ORing together subsets of masks with each pixel value, ncolors * $2^{(nreds+ngreens+nblues)}$ distinct pixel values can be produced. All of these are allocated by the request. However, in the colormap, there are only ncolors * 2^{nreds} independent red entries, ncolors * $2^{ngreens}$ independent green entries, and ncolors * 2^{nblues} independent blue entries. This is true even for `PseudoColor`. When the colormap entry of a pixel value is changed (using `XStoreColors`, `XStoreColor`, or `XStoreNamedColor`), the pixel is decomposed according to the masks, and the corresponding independent entries are updated. `XAllocColorPlanes` returns nonzero if it succeeded or zero if it failed.

`XAllocColorPlanes` can generate `BadColor` and `BadValue` errors.

The `XFreeColors` function frees the cells represented by pixels whose values are in the pixels array. The planes argument should not have any bits set to 1 in common with any of the pixels. The set of all pixels is produced by ORing together subsets of the planes argument with the pixels. The request frees all of these pixels that were allocated by the client (using `XAllocColor`, `XAllocNamedColor`, `XAllocColorCells`, and `XAllocColorPlanes`). Note that freeing an individual pixel obtained from `XAllocColorPlanes` may not actually allow it to be reused until all of its related pixels are also freed.

All specified pixels that are allocated by the client in the colormap are freed, even if one or more pixels produce an error. If a specified pixel is not a valid index into the colormap, a `BadValue` error results. If a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client), a `BadAccess` error results. If more than one pixel is in error, the one that gets reported is arbitrary.

`XFreeColors` can generate `BadAccess`, `BadColor`, and `BadValue` errors.

Diagnostics

`BadAccess`	A client attempted to free a color map entry that it did not already allocate.
`BadAccess`	A client attempted to store into a read-only color map entry.
`BadColor`	A value for a Colormap argument does not name a defined Colormap.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified

for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XCreateColormap(3X11), XQueryColor(3X11), XStoreColors(3X11)
Guide to the Xlib Library

XAllowEvents (3X11)

Name

XAllowEvents – release queued events

Syntax

XAllowEvents(*display*, *event_mode*, *time*)
Display **display*;
int *event_mode*;
Time *time*;

Arguments

display	Specifies the connection to the X server.
event_mode	Specifies the event mode. You can pass `AsyncPointer`, `SyncPointer`, `AsyncKeyboard`, `SyncKeyboard`, `ReplayPointer`, `ReplayKeyboard`, `AsyncBoth`, or `SyncBoth`.
time	Specifies the time. You can pass either a timestamp or `CurrentTime`.

Description

The `XAllowEvents` function releases some queued events if the client has caused a device to freeze. It has no effect if the specified time is earlier than the last-grab time of the most recent active grab for the client or if the specified time is later than the current X server time.

`XAllowEvents` can generate a `BadValue` error.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

Name

XChangeKeyboardControl, XGetKeyboardControl, XAutoRepeatOn, XAutoRepeatOff, XBell, XQueryKeymap – manipulate keyboard settings

Syntax

XChangeKeyboardControl(*display*, *value_mask*, *values*)
Display **display*;
unsigned long *value_mask*;
XKeyboardControl **values*;

XGetKeyboardControl(*display*, *values_return*)
Display **display*;
XKeyboardState **values_return*;

XAutoRepeatOn(*display*)
Display **display*;

XAutoRepeatOff(*display*)
Display **display*;

XBell(*display*, *percent*)
Display **display*;
int *percent*;

XQueryKeymap(*display*, *keys_return*)
Display **display*;
char *keys_return*[32];

Arguments

display	Specifies the connection to the X server.
keys_return	Returns an array of bytes that identifies which keys are pressed down. Each bit represents one key of the keyboard.
percent	Specifies the volume for the bell, which can range from –100 to 100 inclusive.
value_mask	Specifies one value for each bit set to 1 in the mask.
values	Specifies which controls to change. This mask is the bitwise inclusive OR of the valid control mask bits.
values_return	Returns the current keyboard controls in the specified `XKeyboardState` structure.

XChangeKeyboardControl (3X11)

Description

The XChangeKeyboardControl function controls the keyboard characteristics defined by the XKeyboardControl structure. The value_mask argument specifies which values are to be changed.

XChangeKeyboardControl can generate BadMatch and BadValue errors.

The XGetKeyboardControl function returns the current control values for the keyboard to the XKeyboardState structure.

The XAutoRepeatOn function turns on auto-repeat for the keyboard on the specified display.

The XAutoRepeatOff function turns off auto-repeat for the keyboard on the specified display.

The XBell function rings the bell on the keyboard on the specified display, if possible. The specified volume is relative to the base volume for the keyboard. If the value for the percent argument is not in the range –100 to 100 inclusive, a BadValue error results. The volume at which the bell rings when the percent argument is nonnegative is:

base – [(base * percent) / 100] + percent

The volume at which the bell rings when the percent argument is negative is:

base + [(base * percent) / 100]

To change the base volume of the bell, use XChangeKeyboardControl.

XBell can generate a BadValue error.

The XQueryKeymap function returns a bit vector for the logical state of the keyboard, where each bit set to 1 indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N.

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

Diagnostics

BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified

for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XChangeKeyboardMapping(3X11), XSetPointerMapping(3X11)
Guide to the Xlib Library

XChangeKeyboardMapping (3X11)

Name

XChangeKeyboardMapping, XGetKeyboardMapping, XDisplayKeycodes, XSetModifierMapping, XGetModifierMapping, XNewModifiermap, XInsertModifiermapEntry, XDeleteModifiermapEntry, XFreeModifierMap – manipulate keyboard encoding

Syntax

XChangeKeyboardMapping(*display*, *first_keycode*, *keysyms_per_keycode*, *keysyms*, *num_codes*)
Display **display*;
int *first_keycode*;
int *keysyms_per_keycode*;
KeySym **keysyms*;
int *num_codes*;

KeySym *XGetKeyboardMapping(*display*, *first_keycode*, *keycode_count*, *keysyms_per_keycode_return*)
Display **display*;
KeyCode *first_keycode*;
int *keycode_count*;
int **keysyms_per_keycode_return*;

XDisplayKeycodes(*display*, *min_keycodes_return*, *max_keycodes_return*)
Display **display*;
int **min_keycodes_return*, *max_keycodes_return*;

int XSetModifierMapping(*display*, *modmap*)
Display **display*;
XModifierKeymap **modmap*;
XModifierKeymap *XGetModifierMapping(*display*)
Display **display*;

XModifierKeymap *XNewModifiermap(*max_keys_per_mod*)
int *max_keys_per_mod*;

XModifierKeymap *XInsertModifiermapEntry(*modmap*, *keycode_entry*, *modifier*)
XModifierKeymap **modmap*;
KeyCode *keycode_entry*;
int *modifier*;

XModifierKeymap *XDeleteModifiermapEntry(*modmap*, *keycode_entry*, *modifier*)
    XModifierKeymap **modmap*;
    KeyCode *keycode_entry*;
    int *modifier*;

XFreeModifiermap(*modmap*)
    XModifierKeymap **modmap*;

Arguments

display Specifies the connection to the X server.

first_keycode Specifies the first KeyCode that is to be changed or returned.

keycode_count Specifies the number of KeyCodes that are to be returned.

keycode_entry Specifies the KeyCode.

keysyms Specifies a pointer to an array of KeySyms.

keysyms_per_keycode
Specifies the number of KeySyms per KeyCode.

keysyms_per_keycode_return
Returns the number of KeySyms per KeyCode.

max_keys_per_mod
Specifies the number of KeyCode entries preallocated to the modifiers in the map.

max_keycodes_return
Returns the maximum number of KeyCodes.

min_keycodes_return
Returns the minimum number of KeyCodes.

modifier Specifies the modifier.

modmap Specifies a pointer to the `XModifierKeymap` structure.

num_codes Specifies the number of KeyCodes that are to be changed.

Description

The `XChangeKeyboardMapping` function defines the symbols for the specified number of KeyCodes starting with first_keycode. The symbols for KeyCodes outside this range remain unchanged. The number of elements in keysyms must be:

num_codes * keysyms_per_keycode

The specified first_keycode must be greater than or equal to min_keycode returned by XDisplayKeycodes, or a BadValue error results. In addition, the following expression must be less than or equal to max_keycode as returned by XDisplayKeycodes, or a BadValue error results:

first_keycode + num_codes – 1

KeySym number N, counting from zero, for KeyCode K has the following index in keysyms, counting from zero:

(K – first_keycode) * keysyms_per_keycode + N

The specified keysyms_per_keycode can be chosen arbitrarily by the client to be large enough to hold all desired symbols. A special KeySym value of NoSymbol should be used to fill in unused elements for individual KeyCodes. It is legal for NoSymbol to appear in nontrailing positions of the effective list for a KeyCode. XChangeKeyboardMapping generates a MappingNotify event.

There is no requirement that the X server interpret this mapping. It is merely stored for reading and writing by clients.

XChangeKeyboardMapping can generate BadAlloc and BadValue errors.

The XGetKeyboardMapping function returns the symbols for the specified number of KeyCodes starting with first_keycode. The value specified in first_keycode must be greater than or equal to min_keycode as returned by XDisplayKeycodes, or a BadValue error results. In addition, the following expression must be less than or equal to max_keycode as returned by XDisplayKeycodes:

first_keycode + keycode_count – 1

If this is not the case, a BadValue error results. The number of elements in the KeySyms list is:

keycode_count * keysyms_per_keycode_return

KeySym number N, counting from zero, for KeyCode K has the following index in the list, counting from zero:
(K – first_code) * keysyms_per_code_return + N

The X server arbitrarily chooses the keysyms_per_keycode_return value to be large enough to report all requested symbols. A special KeySym value of NoSymbol is used to fill in unused elements for individual KeyCodes. To free the storage returned by XGetKeyboardMapping, use XFree.

XGetKeyboardMapping can generate a BadValue error.

The XDisplayKeycodes function returns the min-keycodes and max-keycodes supported by the specified display. The minimum number of KeyCodes returned is never less than 8, and the maximum number of KeyCodes returned is never greater than 255. Not all KeyCodes in this range are required to have corresponding keys.

The XSetModifierMapping function specifies the KeyCodes of the keys (if any) that are to be used as modifiers. If it succeeds, the X server generates a MappingNotify event, and XSetModifierMapping returns MappingSuccess. X permits at most eight modifier keys. If more than eight are specified in the XModifierKeymap structure, a BadLength error results.

The modifiermap member of the XModifierKeymap structure contains eight sets of max_keypermod KeyCodes, one for each modifier in the order Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5. Only nonzero KeyCodes have meaning in each set, and zero KeyCodes are ignored. In addition, all of the nonzero KeyCodes must be in the range specified by min_keycode and max_keycode in the Display structure, or a BadValue error results. No KeyCode may appear twice in the entire map, or a BadValue error results.

An X server can impose restrictions on how modifiers can be changed, for example, if certain keys do not generate up transitions in hardware, if auto-repeat cannot be disabled on certain keys, or if multiple modifier keys are not supported. If some such restriction is violated, the status reply is MappingFailed, and none of the modifiers are changed. If the new KeyCodes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are in the logically down state, XSetModifierMapping returns MappingBusy, and none of the modifiers is changed.

XSetModifierMapping can generate BadAlloc and BadValue errors.

The XGetModifierMapping function returns a pointer to a newly created XModifierKeymap structure that contains the keys being used as modifiers. The structure should be freed after use by calling XFreeModifiermap. If only zero values appear in the set for any modifier, that modifier is disabled.

The XNewModifiermap function returns a pointer to XModifierKeymap structure for later use.

The XInsertModifiermapEntry function adds the specified KeyCode to the set that controls the specified modifier and returns the resulting XModifierKeymap structure (expanded as needed).

The XDeleteModifiermapEntry function deletes the specified KeyCode from the set that controls the specified modifier and returns a pointer to the resulting XModifierKeymap structure.

The XFreeModifiermap function frees the specified XModifierKeymap structure.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XSetPointerMapping(3X11)
Guide to the Xlib Library

Name

XChangePointerControl, XGetPointerControl – control pointer

Syntax

XChangePointerControl(*display*, *do_accel*, *do_threshold*, *accel_numerator*, *accel_denominator*, *threshold*)
Display **display*;
Bool *do_accel*, *do_threshold*;
int *accel_numerator*, *accel_denominator*;
int *threshold*;

XGetPointerControl(*display*, *accel_numerator_return*, *accel_denominator_return*, *threshold_return*)
Display **display*;
int **accel_numerator_return*, **accel_denominator_return*;
int **threshold_return*;

Arguments

accel_denominator
Specifies the denominator for the acceleration multiplier.

accel_denominator_return
Returns the denominator for the acceleration multiplier.

accel_numerator
Specifies the numerator for the acceleration multiplier.

accel_numerator_return
Returns the numerator for the acceleration multiplier.

display Specifies the connection to the X server.

do_accel Specifies a Boolean value that controls whether the values for the accel_numerator or accel_denominator are used.

do_threshold Specifies a Boolean value that controls whether the value for the threshold is used.

threshold Specifies the acceleration threshold.

threshold_return
Returns the acceleration threshold.

XChangePointerControl (3X11)

Description

The `XChangePointerControl` function defines how the pointing device moves. The acceleration, expressed as a fraction, is a multiplier for movement. For example, specifying 3/1 means the pointer moves three times as fast as normal. The fraction may be rounded arbitrarily by the X server. Acceleration only takes effect if the pointer moves more than threshold pixels at once and only applies to the amount beyond the value in the threshold argument. Setting a value to –1 restores the default. The values of the do_accel and do_threshold arguments must be `True` for the pointer values to be set, or the parameters are unchanged. Negative values (other than –1) generate a `BadValue` error, as does a zero value for the accel_denominator argument.

`XChangePointerControl` can generate a `BadValue` error.

The `XGetPointerControl` function returns the pointer's current acceleration multiplier and acceleration threshold.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

Name

XChangeSaveSet, XAddToSaveSet, XRemoveFromSaveSet – change a client's save set

Syntax

XChangeSaveSet(*display*, *w*, *change_mode*)
Display **display*;
Window *w*;
int *change_mode*;

XAddToSaveSet(*display*, *w*)
Display **display*;
Window *w*;

XRemoveFromSaveSet(*display*, *w*)
Display **display*;
Window *w*;

Arguments

change_mode	Specifies the mode. You can pass `SetModeInsert` or `SetModeDelete`.
display	Specifies the connection to the X server.
w	Specifies the window that you want to add or delete from the client's save-set.

Description

Depending on the specified mode, `XChangeSaveSet` either inserts or deletes the specified window from the client's save-set. The specified window must have been created by some other client, or a `BadMatch` error results.

`XChangeSaveSet` can generate `BadMatch`, `BadValue`, and `BadWindow` errors.

The `XAddToSaveSet` function adds the specified window to the client's save-set. The specified window must have been created by some other client, or a `BadMatch` error results.

`XAddToSaveSet` can generate `BadMatch` and `BadWindow` errors.

The `XRemoveFromSaveSet` function removes the specified window from the client's save-set. The specified window must have been created by some other client, or a `BadMatch` error results.

XRemoveFromSaveSet can generate BadMatch and BadWindow errors.

Diagnostics

BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XReparentWindow(3X11)
Guide to the Xlib Library

Name

XChangeWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap – change window attributes

Syntax

XChangeWindowAttributes(*display*, *w*, *valuemask*, *attributes*)
Display **display*;
Window *w*;
unsigned long *valuemask*;
XSetWindowAttributes **attributes*;

XSetWindowBackground(*display*, *w*, *background_pixel*)
Display **display*;
Window *w*;
unsigned long *background_pixel*;

XSetWindowBackgroundPixmap(*display*, *w*, *background_pixmap*)
Display **display*;
Window *w*;
Pixmap *background_pixmap*;

XSetWindowBorder(*display*, *w*, *border_pixel*)
Display **display*;
Window *w*;
unsigned long *border_pixel*;

XSetWindowBorderPixmap(*display*, *w*, *border_pixmap*)
Display **display*;
Window *w*;
Pixmap *border_pixmap*;

Arguments

attributes Specifies the structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure.

background_pixel
Specifies the pixel that is to be used for the background.

background_pixmap
Specifies the background pixmap, `ParentRelative`, or

	None.
border_pixel	Specifies the entry in the colormap.
border_pixmap	Specifies the border pixmap or CopyFromParent.
display	Specifies the connection to the X server.
valuemask	Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced.
w	Specifies the window.

Description

Depending on the valuemask, the XChangeWindowAttributes function uses the window attributes in the XSetWindowAttributes structure to change the specified window attributes. Changing the background does not cause the window contents to be changed. To repaint the window and its background, use XClearWindow. Setting the border or changing the background such that the border tile origin changes causes the border to be repainted. Changing the background of a root window to None or ParentRelative restores the default background pixmap. Changing the border of a root window to CopyFromParent restores the default border pixmap. Changing the win-gravity does not affect the current position of the window. Changing the backing-store of an obscured window to WhenMapped or Always, or changing the backing-planes, backing-pixel, or save-under of a mapped window may have no immediate effect. Changing the colormap of a window (that is, defining a new map, not changing the contents of the existing map) generates a ColormapNotify event. Changing the colormap of a visible window may have no immediate effect on the screen because the map may not be installed (see XInstallColormap). Changing the cursor of a root window to None restores the default cursor. Whenever possible, you are encouraged to share colormaps.

Multiple clients can select input on the same window. Their event masks are maintained separately. When an event is generated, it is reported to all interested clients. However, only one client at a time can select for SubstructureRedirectMask, ResizeRedirectMask, and ButtonPressMask. If a client attempts to select any of these event masks and some other client has already selected one, a BadAccess error results. There is only one do-not-propagate-mask for a window, not one per client.

XChangeWindowAttributes can generate BadAccess, BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow errors.

The XSetWindowBackground function sets the background of the window to the specified pixel value. Changing the background does not cause the window contents to be changed. XSetWindowBackground uses a pixmap of undefined size filled with the pixel value you passed. If you try to change the background of an InputOnly window, a BadMatch error results.

XSetWindowBackground can generate BadMatch and BadWindow errors.

The XSetWindowBackgroundPixmap function sets the background pixmap of the window to the specified pixmap. The background pixmap can immediately be freed if no further explicit references to it are to be made. If ParentRelative is specified, the background pixmap of the window's parent is used, or on the root window, the default background is restored. If you try to change the background of an InputOnly window, a BadMatch error results. If the background is set to None, the window has no defined background.

XSetWindowBackgroundPixmap can generate BadMatch, BadPixmap, and BadWindow errors.

The XSetWindowBorder function sets the border of the window to the pixel value you specify. If you attempt to perform this on an InputOnly window, a BadMatch error results.

XSetWindowBorder can generate BadMatch and BadWindow errors.

The XSetWindowBorderPixmap function sets the border pixmap of the window to the pixmap you specify. The border pixmap can be freed immediately if no further explicit references to it are to be made. If you specify CopyFromParent, a copy of the parent window's border pixmap is used. If you attempt to perform this on an InputOnly window, a BadMatch error results.

XSetWindowBorderPixmap can generate BadMatch, BadPixmap, and BadWindow errors.

Diagnostics

BadAccess	A client attempted to free a color map entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only color map entry.

`BadColor`	A value for a Colormap argument does not name a defined Colormap.
`BadCursor`	A value for a Cursor argument does not name a defined Cursor.
`BadMatch`	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
`BadMatch`	An `InputOnly` window locks this attribute.
`BadPixmap`	A value for a Pixmap argument does not name a defined Pixmap.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XConfigureWindow(3X11), XCreateWindow(3X11), XDestroyWindow(3X11), XMapWindow(3X11), XRaiseWindow(3X11), XUnmapWindow(3X11)
Guide to the Xlib Library

Name

XClearArea, XClearWindow – clear area or window

Syntax

XClearArea(*display*, *w*, *x*, *y*, *width*, *height*, *exposures*)
Display **display*;
Window *w*;
int *x*, *y*;
unsigned int *width*, *height*;
Bool *exposures*;

XClearWindow(*display*, *w*)
Display **display*;
Window *w*;

Arguments

display	Specifies the connection to the X server.
exposures	Specifies a Boolean value that indicates if `Expose` events are to be generated.
w	Specifies the window.
width *height*	Specify the width and height, which are the dimensions of the rectangle.
x *y*	Specify the x and y coordinates, which are relative to the origin of the window and specify the upper-left corner of the rectangle.

Description

The `XClearArea` function paints a rectangular area in the specified window according to the specified dimensions with the window's background pixel or pixmap. The subwindow-mode effectively is `ClipByChildren`. If width is zero, it is replaced with the current width of the window minus x. If height is zero, it is replaced with the current height of the window minus y. If the window has a defined background tile, the rectangle clipped by any children is filled with this tile. If the window has background `None`, the contents of the window are not changed. In either case, if exposures is `True`, one or more `Expose` events are generated for regions of the rectangle that

are either visible or are being retained in a backing store. If you specify a window whose class is `InputOnly`, a `BadMatch` error results.

`XClearArea` can generate `BadMatch`, `BadValue`, and `BadWindow` errors.

The `XClearWindow` function clears the entire area in the specified window and is equivalent to `XClearArea` (display, w, 0, 0, 0, 0, `False`). If the window has a defined background tile, the rectangle is tiled with a plane-mask of all ones and `GXcopy` function. If the window has background `None`, the contents of the window are not changed. If you specify a window whose class is `InputOnly`, a `BadMatch` error results.

`XClearWindow` can generate `BadMatch` and `BadWindow` errors.

Diagnostics

`BadMatch`	An `InputOnly` window is used as a Drawable.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XCopyArea(3X11)
Guide to the Xlib Library

Name

XConfigureWindow, XMoveWindow, XResizeWindow, XMoveResizeWindow, XSetWindowBorderWidth – configure windows

Syntax

XConfigureWindow(*display*, *w*, *value_mask*, *values*)
Display **display*;
Window *w*;
unsigned int *value_mask*;
XWindowChanges **values*;

XMoveWindow(*display*, *w*, *x*, *y*)
Display **display*;
Window *w*;
int *x*, *y*;

XResizeWindow(*display*, *w*, *width*, *height*)
Display **display*;
Window *w*;
unsigned int *width*, *height*;

XMoveResizeWindow(*display*, *w*, *x*, *y*, *width*, *height*)
Display **display*;
Window *w*;
int *x*, *y*;
unsigned int *width*, *height*;

XSetWindowBorderWidth(*display*, *w*, *width*)
Display **display*;
Window *w*;
unsigned int *width*;

Arguments

display	Specifies the connection to the X server.
value_mask	Specifies which values are to be set using information in the values structure. This mask is the bitwise inclusive OR of the valid configure window values bits.
values	Specifies a pointer to the `XWindowChanges` structure.
w	Specifies the window to be reconfigured, moved, or resized..
width	Specifies the width of the window border.

width
height — Specify the width and height, which are the interior dimensions of the window.

x
y — Specify the x and y coordinates, which define the new location of the top-left pixel of the window's border or the window itself if it has no border or define the new position of the window relative to its parent.

Description

The `XConfigureWindow` function uses the values specified in the `XWindowChanges` structure to reconfigure a window's size, position, border, and stacking order. Values not specified are taken from the existing geometry of the window.

If a sibling is specified without a stack_mode or if the window is not actually a sibling, a `BadMatch` error results. Note that the computations for `BottomIf`, `TopIf`, and `Opposite` are performed with respect to the window's final geometry (as controlled by the other arguments passed to `XConfigureWindow`), not its initial geometry. Any backing store contents of the window, its inferiors, and other newly visible windows are either discarded or changed to reflect the current screen contents (depending on the implementation).

`XConfigureWindow` can generate `BadMatch`, `BadValue`, and `BadWindow` errors.

The `XMoveWindow` function moves the specified window to the specified x and y coordinates, but it does not change the window's size, raise the window, or change the mapping state of the window. Moving a mapped window may or may not lose the window's contents depending on if the window is obscured by nonchildren and if no backing store exists. If the contents of the window are lost, the X server generates `Expose` events. Moving a mapped window generates `Expose` events on any formerly obscured windows.

If the override-redirect flag of the window is `False` and some other client has selected `SubstructureRedirectMask` on the parent, the X server generates a `ConfigureRequest` event, and no further processing is performed. Otherwise, the window is moved.

`XMoveWindow` can generate a `BadWindow` error.

The XResizeWindow function changes the inside dimensions of the specified window, not including its borders. This function does not change the window's upper-left coordinate or the origin and does not restack the window. Changing the size of a mapped window may lose its contents and generate Expose events. If a mapped window is made smaller, changing its size generates Expose events on windows that the mapped window formerly obscured.

If the override-redirect flag of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no further processing is performed. If either width or height is zero, a BadValue error results.

XResizeWindow can generate BadValue and BadWindow errors.

The XMoveResizeWindow function changes the size and location of the specified window without raising it. Moving and resizing a mapped window may generate an Expose event on the window. Depending on the new size and location parameters, moving and resizing a window may generate Expose events on windows that the window formerly obscured.

If the override-redirect flag of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no further processing is performed. Otherwise, the window size and location are changed.

XMoveResizeWindow can generate BadValue and BadWindow errors.

The XSetWindowBorderWidth function sets the specified window's border width to the specified width.

XSetWindowBorderWidth can generate a BadWindow error.

Diagnostics

BadMatch	An InputOnly window is used as a Drawable.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined

Window.

See Also

XChangeWindowAttributes(3X11), XCreateWindow(3X11), XDestroyWindow(3X11), XMapWindow(3X11), XRaiseWindow(3X11), XUnmapWindow(3X11)
Guide to the Xlib Library

Name

XCopyArea, XCopyPlane – copy areas

Syntax

XCopyArea(*display*, *src*, *dest*, *gc*, *src_x*, *src_y*, *width*, *height*, *dest_x*, *dest_y*)
Display **display*;
Drawable *src*, *dest*;
GC *gc*;
int *src_x*, *src_y*;
unsigned int *width*, *height*;
int *dest_x*, *dest_y*;

XCopyPlane(*display*, *src*, *dest*, *gc*, *src_x*, *src_y*, *width*, *height*, *dest_x*, *dest_y*, *plane*)
Display **display*;
Drawable *src*, *dest*;
GC *gc*;
int *src_x*, *src_y*;
unsigned int *width*, *height*;
int *dest_x*, *dest_y*;
unsigned long *plane*;

Arguments

dest_x *dest_y*	Specify the x and y coordinates, which are relative to the origin of the destination rectangle and specify its upper-left corner.
display	Specifies the connection to the X server.
gc	Specifies the GC.
plane	Specifies the bit plane. You must set exactly one bit to 1.
src *dest*	Specify the source and destination rectangles to be combined.
src_x *src_y*	Specify the x and y coordinates, which are relative to the origin of the source rectangle and specify its upper-left corner.
width	

height Specify the width and height, which are the dimensions of both the source and destination rectangles.

Description

The XCopyArea function combines the specified rectangle of src with the specified rectangle of dest. The drawables must have the same root and depth, or a BadMatch error results.

If regions of the source rectangle are obscured and have not been retained in backing store or if regions outside the boundaries of the source drawable are specified, those regions are not copied. Instead, the following occurs on all corresponding destination regions that are either visible or are retained in backing store.

If the destination is a window with a background other than None, corresponding regions of the destination are tiled with that background (with plane-mask of all ones and GXcopy function).

Regardless of tiling or whether the destination is a window or a pixmap, if graphics-exposures is True, then GraphicsExpose events for all corresponding destination regions are generated. If graphics-exposures is True but no GraphicsExpose events are generated, a NoExpose event is generated. Note that by default graphics-exposures is True in new GCs.

This function uses these GC components: function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.

XCopyArea can generate BadDrawable, BadGC, and BadMatch errors.

The XCopyPlane function uses a single bit plane of the specified source rectangle combined with the specified GC to modify the specified rectangle of dest. The drawables must have the same root but need not have the same depth. If the drawables do not have the same root, a BadMatch error results. If plane does not have exactly one bit set to 1 and the values of planes must be less than 2^n, where n is the depth of scr, a BadValue error results.

Effectively, XCopyPlane forms a pixmap of the same depth as the rectangle of dest and with a size specified by the source region. It uses the foreground/background pixels in the GC (foreground everywhere the bit plane in src contains a bit set to 1, background everywhere the bit plane in src contains a bit set to 0) and the equivalent of a CopyArea potocol request is performed with all the same exposure semantics. This can also be thought of as using the specified region of the source bit plane as a stipple with a fill-style of FillOpaqueStippled for filling a rectangular area of the destination.

This function uses these GC components: function, plane-mask, foreground, background, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.

`XCopyPlane` can generate `BadDrawable`, `BadGC`, `BadMatch`, and `BadValue` errors.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

`BadMatch`
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

`BadValue`
: Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XClearArea(3X11)
Guide to the Xlib Library

XCreateColormap (3X11)

Name

XCreateColormap, XCopyColormapAndFree, XFreeColormap, XSetWindowColormap – create, copy, or destroy colormaps

Syntax

Colormap XCreateColormap(*display*, *w*, *visual*, *alloc*)
Display **display*;
Window *w*;
Visual **visual*;
int *alloc*;

Colormap XCopyColormapAndFree(*display*, *colormap*)
Display **display*;
Colormap *colormap*;

XFreeColormap(*display*, *colormap*)
Display **display*;
Colormap *colormap*;

XSetWindowColormap(*display*, *w*, *colormap*)
Display **display*;
Window *w*;
Colormap *colormap*;

Arguments

alloc	Specifies the colormap entries to be allocated. You can pass `AllocNone` or `AllocAll`.
colormap	Specifies the colormap that you want to create, copy, set, or destroy.
display	Specifies the connection to the X server.
visual	Specifies a pointer to a visual type supported on the screen. If the visual type is not one supported by the screen, a `BadMatch` error results.
w	Specifies the window for which you want to create or set a colormap .

Description

The XCreateColormap function creates a colormap of the specified visual type for the screen on which the specified window resides and returns the colormap ID associated with it. Note that the specified window is only used to determine the screen.

The initial values of the colormap entries are undefined for the visual classes GrayScale, PseudoColor, and DirectColor. For StaticGray, StaticColor, and TrueColor, the entries have defined values, but those values are specific to the visual and are not defined by X. For StaticGray, StaticColor, and TrueColor, alloc must be AllocNone, or a BadMatch error results. For the other visual classes, if alloc is AllocNone, the colormap initially has no allocated entries, and clients can allocate them. For information about the visual types, see Section 3.1.

If alloc is AllocAll, the entire colormap is allocated writable. The initial values of all allocated entries are undefined. For GrayScale and PseudoColor, the effect is as if an XAllocColorCells call returned all pixel values from zero to N – 1, where N is the colormap entries value in the specified visual. For DirectColor, the effect is as if an XAllocColorPlanes call returned a pixel value of zero and red_mask, green_mask, and blue_mask values containing the same bits as the corresponding masks in the specified visual. However, in all cases, none of these entries can be freed by using XFreeColors.

XCreateColormap can generate BadAlloc, BadMatch, BadValue, and BadWindow errors.

The XCopyColormapAndFree function creates a colormap of the same visual type and for the same screen as the specified colormap and returns the new colormap ID. It also moves all of the client's existing allocation from the specified colormap to the new colormap with their color values intact and their read-only or writable characteristics intact and frees those entries in the specified colormap. Color values in other entries in the new colormap are undefined.

If the specified colormap was created by the client with alloc set to AllocAll, the new colormap is also created with AllocAll, all color values for all entries are copied from the specified colormap, and then all entries in the specified colormap are freed. If the specified colormap was not created by the client with AllocAll, the allocations to be moved are all those pixels and planes that have been allocated by the client using XAllocColor, XAllocNamedColor, XAllocColorCells, or XAllocColorPlanes and that have not been freed since they were

allocated.

XCopyColormapAndFree can generate BadAlloc and BadColor errors.

The XFreeColormap function deletes the association between the colormap resource ID and the colormap and frees the colormap storage. However, this function has no effect on the default colormap for a screen. If the specified colormap is an installed map for a screen, it is uninstalled (see XUninstallColormap). If the specified colormap is defined as the colormap for a window (by XCreateWindow, XSetWindowColormap, or XChangeWindowAttributes), XFreeColormap changes the colormap associated with the window to None and generates a ColormapNotify event. X does not define the colors displayed for a window with a colormap of None.

XFreeColormap can generate a BadColor error.

The XSetWindowColormap function sets the specified colormap of the specified window. The colormap must have the same visual type as the window, or a BadMatch error results.

XSetWindowColormap can generate BadColor, BadMatch, and BadWindow errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadMatch	An InputOnly window is used as a Drawable.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XAllocColor(3X11), XQueryColor(3X11), XStoreColors(3X11)
Guide to the Xlib Library

XCreateFontCursor (3X11)

Name

XCreateFontCursor, XCreatePixmapCursor, XCreateGlyphCursor – create cursors

Syntax

#include <X11/cursorfont.h>

Cursor XCreateFontCursor(*display*, *shape*)
Display **display*;
unsigned int *shape*;

Cursor XCreatePixmapCursor(*display*, *source*, *mask*, *foreground_color*, *background_color*, *x*, *y*)
Display **display*;
Pixmap *source*;
Pixmap *mask*;
XColor **foreground_color*;
XColor **background_color*;
unsigned int *x*, *y*;

Cursor XCreateGlyphCursor(*display*, *source_font*, *mask_font*, *source_char*, *mask_char*, *foreground_color*, *background_color*)
Display **display*;
Font *source_font*, *mask_font*;
unsigned int *source_char*, *mask_char*;
XColor **foreground_color*;
XColor **background_color*;

Arguments

background_color
Specifies the RGB values for the background of the source.

display Specifies the connection to the X server.

foreground_color
Specifies the RGB values for the foreground of the source.

mask Specifies the cursor's source bits to be displayed or `None`.

mask_char Specifies the glyph character for the mask.

mask_font Specifies the font for the mask glyph or `None`.

shape Specifies the shape of the cursor.

source	Specifies the shape of the source cursor.
source_char	Specifies the character glyph for the source.
source_font	Specifies the font for the source glyph.
x *y*	Specify the x and y coordinates, which indicate the hotspot relative to the source's origin.

Description

X provides a set of standard cursor shapes in a special font named cursor. Applications are encouraged to use this interface for their cursors because the font can be customized for the individual display type. The shape argument specifies which glyph of the standard fonts to use.

The hotspot comes from the information stored in the cursor font. The initial colors of a cursor are a black foreground and a white background (see `XRecolorCursor`).

`XCreateFontCursor` can generate `BadAlloc` and `BadValue` errors.

The `XCreatePixmapCursor` function creates a cursor and returns the cursor ID associated with it. The foreground and background RGB values must be specified using foreground_color and background_color, even if the X server only has a `StaticGray` or `GrayScale` screen. The foreground color is used for the pixels set to 1 in the source, and the background color is used for the pixels set to 0.

Both source and mask, if specified, must have depth one (or a `BadMatch` error results) but can have any root. The Mask argument defines the shape of the cursor. The pixels set to 1 in the mask define which source pixels are displayed, and the pixels set to 0 define which pixels are ignored. If no mask is given, all pixels of the source are displayed. The mask, if present, must be the same size as the pixmap defined by the source argument, or a `BadMatch` error results. The hotspot must be a point within the source, or a `BadMatch` error results.

The components of the cursor can be transformed arbitrarily to meet display limitations. The pixmaps can be freed immediately if no further explicit references to them are to be made. Subsequent drawing in the source or mask pixmap has an undefined effect on the cursor. The X server might or might not make a copy of the pixmap.

`XCreatePixmapCursor` can generate `BadAlloc` and `BadPixmap` errors.

The XCreateGlyphCursor function is similar to XCreatePixmapCursor except that the source and mask bitmaps are obtained from the specified font glyphs. The source_char must be a defined glyph in source_font, or a BadValue error results. If mask_font is given, mask_char must be a defined glyph in mask_font, or a BadValue error results. The mask_font and character are optional.

The origins of the source_char and mask_char (if defined) glyphs are positioned coincidently and define the hotspot. The source_char and mask_char need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask_char is given, all pixels of the source are displayed. You can free the fonts immediately by calling XFreeFont if no further explicit references to them are to be made.

For 2-byte matrix fonts, the 16-bit value should be formed with the byte1 member in the most-significant byte and the byte2 member in the least-significant byte.

XCreateGlyphCursor can generate BadAlloc, BadFont, and BadValue errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadFont	A value for a Font or GContext argument does not name a defined Font.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XDefineCursor(3X11), XRecolorCursor(3X11)
Guide to the Xlib Library

XCreateGC (3X11)

Name

XCreateGC, XCopyGC, XChangeGC, XFreeGC, XGContextFromGC – create or free graphics contexts

Syntax

GC XCreateGC(*display*, *d*, *valuemask*, *values*)
Display **display*;
Drawable *d*;
unsigned long *valuemask*;
XGCValues **values*;

XCopyGC(*display*, *src*, *valuemask*, *dest*)
Display **display*;
GC *src*, *dest*;
unsigned long *valuemask*;

XChangeGC(*display*, *gc*, *valuemask*, *values*)
Display **display*;
GC *gc*;
unsigned long *valuemask*;
XGCValues **values*;

XFreeGC(*display*, *gc*)
Display **display*;
GC *gc*;

GContext XGContextFromGC(*gc*)
GC *gc*;

Arguments

d	Specifies the drawable.
dest	Specifies the destination GC.
display	Specifies the connection to the X server.
gc	Specifies the GC.
src	Specifies the components of the source GC.
valuemask	Specifies which components in the GC are to be set, copied, or changed . This argument is the bitwise inclusive OR of one or more of the valid GC component mask bits.
values	Specifies any values as specified by the valuemask.

Description

The XCreateGC function creates a graphics context and returns a GC. The GC can be used with any destination drawable having the same root and depth as the specified drawable. Use with other drawables results in a BadMatch error.

XCreateGC can generate BadAlloc, BadDrawable, BadFont, BadMatch, BadPixmap, and BadValue errors.

The XCopyGC function copies the specified components from the source GC to the destination GC. The source and destination GCs must have the same root and depth, or a BadMatch error results. The valuemask specifies which component to copy, as for XCreateGC.

XCopyGC can generate BadAlloc, BadGC, and BadMatch errors.

The XChangeGC function changes the components specified by valuemask for the specified GC. The values argument contains the values to be set. The values and restrictions are the same as for XCreateGC. Changing the clip-mask overrides any previous XSetClipRectangles request on the context. Changing the dash-offset or dash-list overrides any previous XSetDashes request on the context. The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered.

XChangeGC can generate BadAlloc, BadFont, BadGC, BadMatch, BadPixmap, and BadValue errors.

The XFreeGC function destroys the specified GC as well as all the associated storage.

XFreeGC can generate a BadGC error.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadFont	A value for a Font or GContext argument does not name a defined Font.
BadGC	A value for a GContext argument does not name a defined GContext.

`BadMatch`	An `InputOnly` window is used as a Drawable.
`BadMatch`	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
`BadPixmap`	A value for a Pixmap argument does not name a defined Pixmap.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XQueryBestSize(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

Name

XCreateImage, XGetPixel, XPutPixel, XSubImage, XAddPixel, XDestroyImage – image utilities

Syntax

XImage *XCreateImage(*display*, *visual*, *depth*, *format*, *offset*, *data*, *width*, *height*, *bitmap_pad*, *bytes_per_line*)
Display **display*;
Visual **visual*;
unsigned int *depth*;
int *format*;
int *offset*;
char **data*;
unsigned int *width*;
unsigned int *height*;
int *bitmap_pad*;
int *bytes_per_line*;

unsigned long XGetPixel(*ximage*, *x*, *y*)
XImage **ximage*;
int *x*;
int *y*;

int XPutPixel(*ximage*, *x*, *y*, *pixel*)
XImage **ximage*;
int *x*;
int *y*;
unsigned long *pixel*;

XImage *XSubImage(*ximage*, *x*, *y*, *subimage_width*, *subimage_height*)
XImage **ximage*;
int *x*;
int *y*;
unsigned int *subimage_width*;
unsigned int *subimage_height*;

XAddPixel(*ximage*, *value*)
XImage **ximage*;
long *value*;

int XDestroyImage(*ximage*)
XImage **ximage*;

Arguments

bitmap_pad	Specifies the quantum of a scanline (8, 16, or 32). In other words, the start of one scanline is separated in client memory from the start of the next scanline by an integer multiple of this many bits.
bytes_per_line	Specifies the number of bytes in the client image between the start of one scanline and the start of the next.
data	Specifies a pointer to the image data.
depth	Specifies the depth of the image.
display	Specifies the connection to the X server.
format	Specifies the format for the image. You can pass `XYBitmap`, `XYPixmap`, or `ZPixmap`.
height	Specifies the height of the image, in pixels.
offset	Specifies the number of pixels to ignore at the beginning of the scanline.
pixel	Specifies the new pixel value.
subimage_height	Specifies the height of the new subimage, in pixels.
subimage_width	Specifies the width of the new subimage, in pixels.
value	Specifies the constant value that is to be added.
visual	Specifies a pointer to the visual.
width	Specifies the width of the image, in pixels.
ximage	Specifies a pointer to the image.
x *y*	Specify the x and y coordinates.

Description

The `XCreateImage` function allocates the memory needed for an `XImage` structure for the specified display but does not allocate space for the image itself. Rather, it initializes the structure byte-order, bit-order, and bitmap-unit values from the display and returns a pointer to the `XImage` structure. The red, green, and blue mask values are defined for Z format images only and are derived from the `Visual` structure passed in. Other values also are passed in. The offset permits the rapid displaying of the image without

requiring each scanline to be shifted into position. If you pass a zero value in bytes_per_line, Xlib assumes that the scanlines are contiguous in memory and calculates the value of bytes_per_line itself.

Note that when the image is created using `XCreateImage`, `XGetImage`, or `XSubImage`, the destroy procedure that the `XDestroyImage` function calls frees both the image structure and the data pointed to by the image structure.

The basic functions used to get a pixel, set a pixel, create a subimage, and add a constant offset to a Z format image are defined in the image object. The functions in this section are really macro invocations of the functions in the image object and are defined in <`X11/Xutil.h`>.

The `XGetPixel` function returns the specified pixel from the named image. The pixel value is returned in normalized format (that is, the least-significant byte of the long is the least-significant byte of the pixel). The image must contain the x and y coordinates.

The `XPutPixel` function overwrites the pixel in the named image with the specified pixel value. The input pixel value must be in normalized format (that is, the least-significant byte of the long is the least-significant byte of the pixel). The image must contain the x and y coordinates.

The `XSubImage` function creates a new image that is a subsection of an existing one. It allocates the memory necessary for the new `XImage` structure and returns a pointer to the new image. The data is copied from the source image, and the image must contain the rectangle defined by x, y, subimage_width, and subimage_height.

The `XAddPixel` function adds a constant value to every pixel in an image. It is useful when you have a base pixel value from allocating color resources and need to manipulate the image to that form.

The `XDestroyImage` function deallocates the memory associated with the `XImage` structure.

See Also

XPutImage(3X11)
Guide to the Xlib Library

XCreatePixmap (3X11)

Name

XCreatePixmap, XFreePixmap – create or destroy pixmaps

Syntax

Pixmap XCreatePixmap(*display*, *d*, *width*, *height*, *depth*)
Display **display*;
Drawable *d*;
unsigned int *width*, *height*;
unsigned int *depth*;

XFreePixmap(*display*, *pixmap*)
Display **display*;
Pixmap *pixmap*;

Arguments

d	Specifies which screen the pixmap is created on.
depth	Specifies the depth of the pixmap.
display	Specifies the connection to the X server.
pixmap	Specifies the pixmap.
width *height*	Specify the width and height, which define the dimensions of the pixmap.

Description

The `XCreatePixmap` function creates a pixmap of the width, height, and depth you specified and returns a pixmap ID that identifies it. It is valid to pass an `InputOnly` window to the drawable argument. The width and height arguments must be nonzero, or a `BadValue` error results. The depth argument must be one of the depths supported by the screen of the specified drawable, or a `BadValue` error results.

The server uses the specified drawable to determine on which screen to create the pixmap. The pixmap can be used only on this screen and only with other drawables of the same depth (see `XCopyPlane` for an exception to this rule). The initial contents of the pixmap are undefined.

`XCreatePixmap` can generate `BadAlloc`, `BadDrawable`, and `BadValue` errors.

The XFreePixmap function first deletes the association between the pixmap ID and the pixmap. Then, the X server frees the pixmap storage when there are no references to it. The pixmap should never be referenced again.

XFreePixmap can generate a BadPixmap error.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

XCreateRegion (3X11)

Name

XCreateRegion, XSetRegion, XDestroyRegion – create or destroy regions

Syntax

Region XCreateRegion()

XSetRegion(*display*, *gc*, *r*)
Display **display*;
GC *gc*;
Region *r*;

XDestroyRegion(*r*)
Region *r*;

Arguments

display	Specifies the connection to the X server.
gc	Specifies the GC.
r	Specifies the region.

Description

The `XCreateRegion` function creates a new empty region.

The `XSetRegion` function sets the clip-mask in the GC to the specified region. Once it is set in the GC, the region can be destroyed.

The `XDestroyRegion` function deallocates the storage associated with a specified region.

See Also

XEmptyRegion(3X11), XIntersectRegion(3X11)
Guide to the Xlib Library

Name

XCreateWindow, XCreateSimpleWindow – create windows

Syntax

Window XCreateWindow(*display*, *parent*, *x*, *y*, *width*, *height*, *border_width*, *depth*, *class*, *visual*, *valuemask*, *attributes*)
Display **display*;
Window *parent*;
int *x*, *y*;
unsigned int *width*, *height*;
unsigned int *border_width*;
int *depth*;
unsigned int *class*;
Visual **visual*
unsigned long *valuemask*;
XSetWindowAttributes **attributes*;

Window XCreateSimpleWindow(*display*, *parent*, *x*, *y*, *width*, *height*, *border_width*, *border*, *background*)
Display **display*;
Window *parent*;
int *x*, *y*;
unsigned int *width*, *height*;
unsigned int *border_width*;
unsigned long *border*;
unsigned long *background*;

Arguments

attributes	Specifies the structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure.
background	Specifies the background pixel value of the window.
border	Specifies the border pixel value of the window.
border_width	Specifies the width of the created window's border in pixels.
class	Specifies the created window's class. You can pass `InputOutput`, `InputOnly`, or `CopyFromParent`. A

	class of `CopyFromParent` means the class is taken from the parent.
depth	Specifies the window's depth. A depth of `CopyFromParent` means the depth is taken from the parent.
display	Specifies the connection to the X server.
parent	Specifies the parent window.
valuemask	Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced.
visual	Specifies the visual type. A visual of `CopyFromParent` means the visual type is taken from the parent.
width *height*	Specify the width and height, which are the created window's inside dimensions and do not include the created window's borders.
x *y*	Specify the x and y coordinates, which are the top-left outside corner of the window's borders and are relative to the inside of the parent window's borders.

Description

The `XCreateWindow` function creates an unmapped subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a `CreateNotify` event. The created window is placed on top in the stacking order with respect to siblings.

The border_width for an `InputOnly` window must be zero, or a `BadMatch` error results. For class `InputOutput`, the visual type and depth must be a combination supported for the screen, or a `BadMatch` error results. The depth need not be the same as the parent, but the parent must not be a window of class `InputOnly`, or a `BadMatch` error results. For an `InputOnly` window, the depth must be zero, and the visual must be one supported by the screen. If either condition is not met, a `BadMatch` error results. The parent window, however, may have any depth and class. If you specify any invalid window attribute for a window, a `BadMatch` error results.

The created window is not yet displayed (mapped) on the user's display. To display the window, call XMapWindow. The new window initially uses the same cursor as its parent. A new cursor can be defined for the new window by calling XDefineCursor. The window will not be visible on the screen unless it and all of its ancestors are mapped and it is not obscured by any of its ancestors.

XCreateWindow can generate BadAlloc BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow errors.

The XCreateSimpleWindow function creates an unmapped InputOutput subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a CreateNotify event. The created window is placed on top in the stacking order with respect to siblings. Any part of the window that extends outside its parent window is clipped. The border_width for an InputOnly window must be zero, or a BadMatch error results. XCreateSimpleWindow inherits its depth, class, and visual from its parent. All other window attributes, except background and border, have their default values.

XCreateSimpleWindow can generate BadAlloc, BadMatch, BadValue, and BadWindow errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadMatch	The values do not exist for an InputOnly window.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of

alternatives can generate this error.

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

XChangeWindowAttributes(3X11), XConfigureWindow(3X11), XDestroyWindow(3X11), XMapWindow(3X11), XRaiseWindow(3X11), XUnmapWindow(3X11)
Guide to the Xlib Library

Name

XDefineCursor, XUndefineCursor – define cursors

Syntax

XDefineCursor(*display*, *w*, *cursor*)
Display **display*;
Window *w*;
Cursor *cursor*;

XUndefineCursor(*display*, *w*)
Display **display*;
Window *w*;

Arguments

cursor	Specifies the cursor that is to be displayed or `None`.
display	Specifies the connection to the X server.
w	Specifies the window.

Description

If a cursor is set, it will be used when the pointer is in the window. If the cursor is `None`, it is equivalent to `XUndefineCursor`.

`XDefineCursor` can generate `BadCursor` and `BadWindow` errors.

The `XUndefineCursor` undoes the effect of a previous `XDefineCursor` for this window. When the pointer is in the window, the parent's cursor will now be used. On the root window, the default cursor is restored.

`XUndefineCursor` can generate a `BadWindow` error.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadCursor`	A value for a Cursor argument does not name a defined Cursor.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XCreateFontCursor(3X11), XRecolorCursor(3X11)
Guide to the Xlib Library

Name

XDestroyWindow, XDestroySubwindows – destroy windows

Syntax

XDestroyWindow(*display*, *w*)
Display **display*;
Window *w*;

XDestroySubwindows(*display*, *w*)
Display **display*;
Window *w*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.

Description

The `XDestroyWindow` function destroys the specified window as well as all of its subwindows and causes the X server to generate a `DestroyNotify` event for each window. The window should never be referenced again. If the window specified by the w argument is mapped, it is unmapped automatically. The ordering of the `DestroyNotify` events is such that for any given window being destroyed, `DestroyNotify` is generated on any inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is not otherwise constrained. If the window you specified is a root window, no windows are destroyed. Destroying a mapped window will generate `Expose` events on other windows that were obscured by the window being destroyed.

`XDestroyWindow` can generate a `BadWindow` error.

The `XDestroySubwindows` function destroys all inferior windows of the specified window, in bottom-to-top stacking order. It causes the X server to generate a `DestroyNotify` event for each window. If any mapped subwindows were actually destroyed, `XDestroySubwindows` causes the X server to generate `Expose` events on the specified window. This is much more efficient than deleting many windows one at a time because much of the work need be performed only once for all of the windows, rather than for each window. The subwindows should never be referenced again.

`XDestroySubwindows` can generate a `BadWindow` error.

Diagnostics

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

XChangeWindowAttributes(3X11), XConfigureWindow(3X11), XCreateWindow(3X11), XMapWindow(3X11), XRaiseWindow(3X11), XUnmapWindow(3X11)
Guide to the Xlib Library

Name

XDrawArc, XDrawArcs – draw arcs

Syntax

XDrawArc(*display*, *d*, *gc*, *x*, *y*, *width*, *height*, *angle1*, *angle2*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;
int *angle1*, *angle2*;

XDrawArcs(*display*, *d*, *gc*, *arcs*, *narcs*)
Display **display*;
Drawable *d*;
GC *gc*;
XArc **arcs*;
int *narcs*;

Arguments

angle1	Specifies the start of the arc relative to the three-o'clock position from the center, in units of degrees * 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.
arcs	Specifies a pointer to an array of arcs.
d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
narcs	Specifies the number of arcs in the array.
width *height*	Specify the width and height, which are the major and minor axes of the arc.
x *y*	Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the bounding rectangle.

XDrawArc (3X11)

Description

`XDrawArc` draws a single circular or elliptical arc, and `XDrawArcs` draws multiple circular or elliptical arcs. Each arc is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height. Positive angles indicate counterclockwise motion, and negative angles indicate clockwise motion. If the magnitude of angle2 is greater than 360 degrees, `XDrawArc` or `XDrawArcs` truncates it to 360 degrees.

For an arc specified as [*x*, *y*, *width*, *height*, *angle*1, *angle*2], the origin of the major and minor axes is at $[x+\frac{width}{2}, y+\frac{height}{2}]$, and the infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at $[x, y+\frac{height}{2}]$ and $[x+width, y+\frac{height}{2}]$ and intersects the vertical axis at $[x+\frac{width}{2}, y]$ and $[x+\frac{width}{2}, y+height]$. These coordinates can be fractional and so are not truncated to discrete coordinates. The path should be defined by the ideal mathematical path. For a wide line with line-width lw, the bounding outlines for filling are given by the two infinitely thin paths consisting of all points whose perpendicular distance from the path of the circle/ellipse is equal to lw/2 (which may be a fractional value). The cap-style and join-style are applied the same as for a line corresponding to the tangent of the circle/ellipse at the endpoint.

For an arc specified as [*x*, *y*, *width*, *height*, *angle*1, *angle*2], the angles must be specified in the effectively skewed coordinate system of the ellipse (for a circle, the angles and coordinate systems are identical). The relationship between these angles and angles expressed in the normal coordinate system of the screen (as measured with a protractor) is as follows:

$$\text{skewed-angle} = atan\left(\tan(\text{normal-angle})*\frac{width}{height}\right)+adjust$$

The skewed-angle and normal-angle are expressed in radians (rather than in degrees scaled by 64) in the range [0, 2π] and where atan returns a value in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$ and adjust is:

0	for normal-angle in the range $[0, \frac{\pi}{2}]$
π	for normal-angle in the range $[\frac{\pi}{2}, \frac{3\pi}{2}]$
2π	for normal-angle in the range $[\frac{3\pi}{2}, 2\pi]$

For any given arc, XDrawArc and XDrawArcs do not draw a pixel more than once. If two arcs join correctly and if the line-width is greater than zero and the arcs intersect, XDrawArc and XDrawArcs do not draw a pixel more than once. Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times. Specifying an arc with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

If the last point in one arc coincides with the first point in the following arc, the two arcs will join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs will join correctly. By specifying one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio.

Both functions use these GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawArc XDrawArcs and can generate BadDrawable, BadGC, and BadMatch errors.

Diagnostics

BadDrawable
: A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC
: A value for a GContext argument does not name a defined GContext.

BadMatch
: An InputOnly window is used as a Drawable.

BadMatch
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

See Also

XDrawLine(3X11), XDrawPoint(3X11), XDrawRectangle(3X11)
Guide to the Xlib Library

XDrawImageString (3X11)

Name

XDrawImageString, XDrawImageString16 – draw image text

Syntax

XDrawImageString(*display*, *d*, *gc*, *x*, *y*, *string*, *length*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
char **string*;
int *length*;

XDrawImageString16(*display*, *d*, *gc*, *x*, *y*, *string*, *length*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
XChar2b **string*;
int *length*;

Arguments

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
length	Specifies the number of characters in the string argument.
string	Specifies the character string.
x *y*	Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.

Description

The `XDrawImageString16` function is similar to `XDrawImageString` except that it uses 2-byte or 16-bit characters. Both functions also use both the foreground and background pixels of the GC in the destination.

The effect is first to fill a destination rectangle with the background pixel defined in the GC and then to paint the text with the foreground pixel. The upper-left corner of the filled rectangle is at:

[x, y – font-ascent]

The width is:

overall-width

The height is:

font-ascent + font-descent

The overall-width, font-ascent, and font-descent are as would be returned by `XQueryTextExtents` using gc and string. The function and fill-style defined in the GC are ignored for these functions. The effective function is `GXcopy`, and the effective fill-style is `FillSolid`.

For fonts defined with 2-byte matrix indexing and used with `XDrawImageString`, each byte is used as a byte2 with a byte1 of zero.

Both functions use these GC components: plane-mask, foreground, background, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

`XDrawImageString` `XDrawImageString16` and can generate `BadDrawable`, `BadGC`, and `BadMatch` errors.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

`BadMatch`
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

See Also

XDrawString(3X11), XDrawText(3X11)
Guide to the Xlib Library

XDrawLine (3X11)

Name

XDrawLine, XDrawLines, XDrawSegments – draw lines and polygons

Syntax

XDrawLine(*display*, *d*, *gc*, *x1*, *y1*, *x2*, *y2*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x1*, *y1*, *x2*, *y2*;

XDrawLines(*display*, *d*, *gc*, *points*, *npoints*, *mode*)
Display **display*;
Drawable *d*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *mode*;

XDrawSegments(*display*, *d*, *gc*, *segments*, *nsegments*)
Display **display*;
Drawable *d*;
GC *gc*;
XSegment **segments*;
int *nsegments*;

Arguments

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
mode	Specifies the coordinate mode. You can pass `CoordModeOrigin` or `CoordModePrevious`.
npoints	Specifies the number of points in the array.
nsegments	Specifies the number of segments in the array.
points	Specifies a pointer to an array of points.
segments	Specifies a pointer to an array of segments.
x1	
y1	

x2
y2 Specify the points (x1, y1) and (x2, y2) to be connected.

Description

The `XDrawLine` function uses the components of the specified GC to draw a line between the specified set of points (x1, y1) and (x2, y2). It does not perform joining at coincident endpoints. For any given line, `XDrawLine` does not draw a pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times.

The `XDrawLines` function uses the components of the specified GC to draw npoints–1 lines between each pair of points (point[i], point[i+1]) in the array of `XPoint` structures. It draws the lines in the order listed in the array. The lines join correctly at all intermediate points, and if the first and last points coincide, the first and last lines also join correctly. For any given line, `XDrawLines` does not draw a pixel more than once. If thin (zero line-width) lines intersect, the intersecting pixels are drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire `PolyLine` protocol request were a single, filled shape. `CoordModeOrigin` treats all coordinates as relative to the origin, and `CoordModePrevious` treats all coordinates after the first as relative to the previous point.

The `XDrawSegments` function draws multiple, unconnected lines. For each segment, `XDrawSegments` draws a line between (x1, y1) and (x2, y2). It draws the lines in the order listed in the array of `XSegment` structures and does not perform joining at coincident endpoints. For any given line, `XDrawSegments` does not draw a pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times.

All three functions use these GC components: function, plane-mask, line-width, line-style, cap-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. The `XDrawLines` function also uses the join-style GC component. All three functions also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

`XDrawLine` `XDrawLines` `XDrawSegments` and can generate `BadDrawable`, `BadGC`, and `BadMatch` errors. `XDrawLines` can also generate a `BadValue` error.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

`BadMatch`
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

`BadValue`
: Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XDrawArc(3X11), XDrawPoint(3X11), XDrawRectangle(3X11)
Guide to the Xlib Library

Name

XDrawPoint, XDrawPoints – draw points

Syntax

XDrawPoint(*display*, *d*, *gc*, *x*, *y*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;

XDrawPoints(*display*, *d*, *gc*, *points*, *npoints*, *mode*)
Display **display*;
Drawable *d*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *mode*;

Arguments

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
mode	Specifies the coordinate mode. You can pass `CoordModeOrigin` or `CoordModePrevious`.
npoints	Specifies the number of points in the array.
points	Specifies a pointer to an array of points.
x *y*	Specify the x and y coordinates where you want the point drawn.

Description

The `XDrawPoint` function uses the foreground pixel and function components of the GC to draw a single point into the specified drawable; `XDrawPoints` draws multiple points this way. `CoordModeOrigin` treats all coordinates as relative to the origin, and `CoordModePrevious` treats all coordinates after the first as relative to the previous point. `XDrawPoints` draws the points in the order listed in the array.

Both functions use these GC components: function, plane-mask, foreground, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

`XDrawPoint` can generate `BadDrawable`, `BadGC`, and `BadMatch` errors. `XDrawPoints` can generate `BadDrawable`, `BadGC`, `BadMatch`, and `BadValue` errors.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

`BadMatch`
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

`BadValue`
: Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XDrawArc(3X11), XDrawLine(3X11), XDrawRectangle(3X11)
Guide to the Xlib Library

Name

XDrawRectangle, XDrawRectangles – draw rectangles

Syntax

XDrawRectangle(*display*, *d*, *gc*, *x*, *y*, *width*, *height*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;

XDrawRectangles(*display*, *d*, *gc*, *rectangles*, *nrectangles*)
Display **display*;
Drawable *d*;
GC *gc*;
XRectangle *rectangles*[];
int *nrectangles*;

Arguments

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
nrectangles	Specifies the number of rectangles in the array.
rectangles	Specifies a pointer to an array of rectangles.
width *height*	Specify the width and height, which specify the dimensions of the rectangle.
x *y*	Specify the x and y coordinates, which specify the upper-left corner of the rectangle.

Description

The `XDrawRectangle` and `XDrawRectangles` functions draw the outlines of the specified rectangle or rectangles as if a five-point `PolyLine` protocol request were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height] [x,y]

For the specified rectangle or rectangles, these functions do not draw a pixel more than once. `XDrawRectangles` draws the rectangles in the order listed in the array. If rectangles intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, line-width, line-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

`XDrawRectangle` `XDrawRectangles` and can generate `BadDrawable`, `BadGC`, and `BadMatch` errors.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

`BadMatch`
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

See Also

XDrawArc(3X11), XDrawLine(3X11), XDrawPoint(3X11)
Guide to the Xlib Library

Name

XDrawString, XDrawString16 – draw text characters

Syntax

XDrawString(*display*, *d*, *gc*, *x*, *y*, *string*, *length*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
char **string*;
int *length*;

XDrawString16(*display*, *d*, *gc*, *x*, *y*, *string*, *length*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
XChar2b **string*;
int *length*;

Arguments

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
length	Specifies the number of characters in the string argument.
string	Specifies the character string.
x *y*	Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.

Description

Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. For fonts defined with 2-byte matrix indexing and used with `XDrawString16`, each byte is used as a byte2 with a byte1 of zero.

Both functions use these GC components: function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

`XDrawString` `XDrawString16` and can generate `BadDrawable`, `BadGC`, and `BadMatch` errors.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

`BadMatch`
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

See Also

XDrawImageString(3X11), XDrawText(3X11)
Guide to the Xlib Library

Name

XDrawText, XDrawText16 – draw polytext text

Syntax

XDrawText(*display*, *d*, *gc*, *x*, *y*, *items*, *nitems*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
XTextItem **items*;
int *nitems*;

XDrawText16(*display*, *d*, *gc*, *x*, *y*, *items*, *nitems*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
XTextItem16 **items*;
int *nitems*;

Arguments

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
items	Specifies a pointer to an array of text items.
nitems	Specifies the number of text items in the array.
x *y*	Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.

Description

The `XDrawText16` function is similar to `XDrawText` except that it uses 2-byte or 16-bit characters. Both functions allow complex spacing and font shifts between counted strings.

Each text item is processed in turn. A font member other than `None` in an item causes the font to be stored in the GC and used for subsequent text. A text element delta specifies an additional change in the position along the x axis before the string is drawn. The delta is always added to the character origin and is not dependent on any characteristics of the font. Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. If a text item generates a `BadFont` error, the previous text items may have been drawn.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each `XChar2b` structure is interpreted as a 16-bit number with byte1 as the most-significant byte.

Both functions use these GC components: function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

`XDrawText` `XDrawText16` and can generate `BadDrawable`, `BadFont`, `BadGC`, and `BadMatch` errors.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadFont`
: A value for a Font or GContext argument does not name a defined Font.

`BadGC`
: A value for a GContext argument does not name a defined GContext.

`BadMatch`
: An `InputOnly` window is used as a Drawable.

See Also

XDrawImageString(3X11), XDrawString(3X11)
Guide to the Xlib Library

Name

XEmptyRegion, XEqualRegion, XPointInRegion, XRectInRegion – determine if regions are empty or equal

Syntax

Bool XEmptyRegion(*r*)
    Region *r*;

Bool XEqualRegion(*r1*, *r2*)
    Region *r1*, *r2*;

Bool XPointInRegion(*r*, *x*, *y*)
    Region *r*;
    int *x*, *y*;

int XRectInRegion(*r*, *x*, *y*, *width*, *height*)
    Region *r*;
    int *x*, *y*;
    unsigned int *width*, *height*;

Arguments

r	Specifies the region.
r1 *r2*	Specify the two regions.
width *height*	Specify the width and height, which define the rectangle.
x *y*	Specify the x and y coordinates, which define the point or the coordinates of the upper-left corner of the rectangle.

Description

The `XEmptyRegion` function returns `True` if the region is empty.

The `XEqualRegion` function returns `True` if the two regions have the same offset, size, and shape.

The `XPointInRegion` function returns `True` if the point (x, y) is contained in the region r.

XEmptyRegion (3X11)

The `XRectInRegion` function returns `RectangleIn` if the rectangle is entirely in the specified region, `RectangleOut` if the rectangle is entirely out of the specified region, and `RectanglePart` if the rectangle is partially in the specified region.

See Also

XCreateRegion(3X11), XIntersectRegion(3X11)
Guide to the Xlib Library

Name

XFillRectangle, XFillRectangles, XFillPolygon, XFillArc, XFillArcs – fill rectangles, polygons, or arcs

Syntax

XFillRectangle(*display*, *d*, *gc*, *x*, *y*, *width*, *height*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;

XFillRectangles(*display*, *d*, *gc*, *rectangles*, *nrectangles*)
Display **display*;
Drawable *d*;
GC *gc*;
XRectangle **rectangles*;
int *nrectangles*;

XFillPolygon(*display*, *d*, *gc*, *points*, *npoints*, *shape*, *mode*)
Display **display*;
Drawable *d*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *shape*;
int *mode*;

XFillArc(*display*, *d*, *gc*, *x*, *y*, *width*, *height*, *angle1*, *angle2*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x*, *y*;
unsigned int *width*, *height*;
int *angle1*, *angle2*;

XFillArcs(*display*, *d*, *gc*, *arcs*, *narcs*)
Display **display*;
Drawable *d*;
GC *gc*;
XArc **arcs*;
int *narcs*;

Arguments

angle1	Specifies the start of the arc relative to the three-o'clock position from the center, in units of degrees * 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.
arcs	Specifies a pointer to an array of arcs.
d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
mode	Specifies the coordinate mode. You can pass `CoordModeOrigin` or `CoordModePrevious`.
narcs	Specifies the number of arcs in the array.
npoints	Specifies the number of points in the array.
nrectangles	Specifies the number of rectangles in the array.
points	Specifies a pointer to an array of points.
rectangles	Specifies a pointer to an array of rectangles.
shape	Specifies a shape that helps the server to improve performance. You can pass `Complex`, `Convex`, or `Nonconvex`.
width *height*	Specify the width and height, which are the dimensions of the rectangle to be filled or the major and minor axes of the arc.
x *y*	Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the rectangle.

Description

The `XFillRectangle` and `XFillRectangles` functions fill the specified rectangle or rectangles as if a four-point `FillPolygon` protocol request were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height]

Each function uses the x and y coordinates, width and height dimensions, and GC you specify.

`XFillRectangles` fills the rectangles in the order listed in the array. For any given rectangle, `XFillRectangle` and `XFillRectangles` do not draw a pixel more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

`XFillRectangle` `XFillRectangles` and can generate `BadDrawable`, `BadGC`, and `BadMatch` errors.

`XFillPolygon` fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. `XFillPolygon` does not draw a pixel of the region more than once. `CoordModeOrigin` treats all coordinates as relative to the origin, and `CoordModePrevious` treats all coordinates after the first as relative to the previous point.

Depending on the specified shape, the following occurs:

- If shape is `Complex`, the path may self-intersect.
- If shape is `Convex`, the path is wholly convex. If known by the client, specifying `Convex` can improve performance. If you specify `Convex` for a path that is not convex, the graphics results are undefined.
- If shape is `Nonconvex`, the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifying `Nonconvex` instead of `Complex` may improve performance. If you specify `Nonconvex` for a self-intersecting path, the graphics results are undefined.

The fill-rule of the GC controls the filling behavior of self-intersecting polygons.

This function uses these GC components: function, plane-mask, fill-style, fill-rule, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. It also uses these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

`XFillPolygon` can generate `BadDrawable`, `BadGC`, `BadMatch`, and `BadValue` errors.

For each arc, XFillArc or XFillArcs fills the region closed by the infinitely thin path described by the specified arc and, depending on the arc-mode specified in the GC, one or two line segments. For ArcChord, the single line segment joining the endpoints of the arc is used. For ArcPieSlice, the two line segments joining the endpoints of the arc with the center point are used. XFillArcs fills the arcs in the order listed in the array. For any given arc, XFillArc and XFillArcs do not draw a pixel more than once. If regions intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, fill-style, arc-mode, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillArc XFillArcs and can generate BadDrawable, BadGC, and BadMatch errors.

Diagnostics

BadDrawable
: A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC
: A value for a GContext argument does not name a defined GContext.

BadMatch
: An InputOnly window is used as a Drawable.

BadMatch
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue
: Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XDrawArc(3X11), XDrawRectangle(3X11)
Guide to the Xlib Library

Name

XFlush, XSync, XEventsQueued, XPending – handle the output buffer or event queue

Syntax

XFlush(*display*)
    Display **display*;

XSync(*display*, *discard*)
    Display **display*;
    Bool *discard*;

int XEventsQueued(*display*, *mode*)
    Display **display*;
    int *mode*;

int XPending(*display*)
    Display **display*;

Arguments

discard	Specifies a Boolean value that indicates whether `XSync` discards all events on the event queue.
display	Specifies the connection to the X server.
mode	Specifies the mode. You can pass `QueuedAlready`, `QueuedAfterFlush`, or `QueuedAfterReading`.

Description

The `XFlush` function flushes the output buffer. Most client applications need not use this function because the output buffer is automatically flushed as needed by calls to `XPending`, `XNextEvent`, and `XWindowEvent`. Events generated by the server may be enqueued into the library's event queue.

The `XSync` function flushes the output buffer and then waits until all requests have been received and processed by the X server. Any errors generated must be handled by the error handler. For each error event received by Xlib, `XSync` calls the client application's error handling routine (see section 8.12.2). Any events generated by the server are enqueued into the library's event queue.

Finally, if you passed `False`, `XSync` does not discard the events in the queue. If you passed `True`, `XSync` discards all events in the queue, including those events that were on the queue before `XSync` was called. Client applications seldom need to call `XSync`.

If mode is `QueuedAlready`, `XEventsQueued` returns the number of events already in the event queue (and never performs a system call). If mode is `QueuedAfterFlush`, `XEventsQueued` returns the number of events already in the queue if the number is nonzero. If there are no events in the queue, `XEventsQueued` flushes the output buffer, attempts to read more events out of the application's connection, and returns the number read. If mode is `QueuedAfterReading`, `XEventsQueued` returns the number of events already in the queue if the number is nonzero. If there are no events in the queue, `XEventsQueued` attempts to read more events out of the application's connection without flushing the output buffer and returns the number read.

`XEventsQueued` always returns immediately without I/O if there are events already in the queue. `XEventsQueued` with mode `QueuedAfterFlush` is identical in behavior to `XPending`. `XEventsQueued` with mode `QueuedAlready` is identical to the `XQLength` function.

The `XPending` function returns the number of events that have been received from the X server but have not been removed from the event queue. `XPending` is identical to `XEventsQueued` with the mode `QueuedAfterFlush` specified.

See Also

XIfEvent(3X11), XNextEvent(3X11), XPutBackEvent(3X11)
Guide to the Xlib Library

Name

XFree, XNoOp – free client data

Syntax

XFree(*data*)
char **data*;

XNoOp(*display*)
Display **display*;

Arguments

display	Specifies the connection to the X server.
data	Specifies a pointer to the data that is to be freed.

Description

The `XFree` function is a general-purpose Xlib routine that frees the specified data. You must use it to free any objects that were allocated by Xlib.

The `XNoOp` function sends a `NoOperation` protocol request to the X server, thereby exercising the connection.

See Also

Guide to the Xlib Library

XGetDefault (3X11)

Name

XGetDefault, XResourceManagerString – get X program defaults

Syntax

char *XGetDefault(*display*, *program*, *option*)
Display **display*;
char **program*;
char **option*;

char *XResourceManagerString(*display*)
Display **display*;

Arguments

display	Specifies the connection to the X server.
option	Specifies the option name.
program	Specifies the program name for the Xlib defaults (usually argv[0] of the main program).

Description

The `XGetDefault` function returns the value NULL if the option name specified in this argument does not exist for the program. The strings returned by `XGetDefault` are owned by Xlib and should not be modified or freed by the client.

The `XResourceManagerString` returns the RESOURCE_MANAGER property from the server's root window of screen zero, which was returned when the connection was opened using `XOpenDisplay`.

See Also

XrmGetSearchList(3X11)
Guide to the Xlib Library

Name

XGetVisualInfo, XMatchVisualInfo, XVisualIDFromVisual – obtain visual information

Syntax

XVisualInfo *XGetVisualInfo(*display*, *vinfo_mask*, *vinfo_template*, *nitems_return*)
Display **display*;
long *vinfo_mask*;
XVisualInfo **vinfo_template*;
int **nitems_return*;

Status XMatchVisualInfo(*display*, *screen*, *depth*, *class*, *vinfo_return*)
Display **display*;
int *screen*;
int *depth*;
int *class*;
XVisualInfo **vinfo_return*;

VisualID XVisualIDFromVisual(*visual*)
Visual **visual*;

Arguments

class	Specifies the class of the screen.
depth	Specifies the depth of the screen.
display	Specifies the connection to the X server.
nitems_return	Returns the number of matching visual structures.
screen	Specifies the screen.
visual	Specifies the visual type.
vinfo_mask	Specifies the visual mask value.
vinfo_return	Returns the matched visual information.
vinfo_template	Specifies the visual attributes that are to be used in matching the visual structures.

XGetVisualInfo (3X11)

Description

The `XGetVisualInfo` function returns a list of visual structures that match the attributes specified by vinfo_template. If no visual structures match the template using the specified vinfo_mask, `XGetVisualInfo` returns a NULL. To free the data returned by this function, use `XFree`.

The `XMatchVisualInfo` function returns the visual information for a visual that matches the specified depth and class for a screen. Because multiple visuals that match the specified depth and class can exist, the exact visual chosen is undefined. If a visual is found, `XMatchVisualInfo` returns nonzero and the information on the visual to vinfo_return. Otherwise, when a visual is not found, `XMatchVisualInfo` returns zero.

The `XVisualIDFromVisual` function returns the visual ID for the specified visual type.

See Also

Guide to the Xlib Library

Name

XGetWindowAttributes, XGetGeometry – get current window attribute or geometry

Syntax

Status XGetWindowAttributes(*display*, *w*, *window_attributes_return*)
Display **display*;
Window *w*;
XWindowAttributes **window_attributes_return*;

Status XGetGeometry(*display*, *d*, *root_return*, *x_return*, *y_return*, *width_return*, *height_return*, *border_width_return*, *depth_return*)
Display **display*;
Drawable *d*;
Window **root_return*;
int **x_return*, **y_return*;
unsigned int **width_return*, **height_return*;
unsigned int **border_width_return*;
unsigned int **depth_return*;

Arguments

border_width_return
Returns the border width in pixels.

d Specifies the drawable, which can be a window or a pixmap.

depth_return Returns the depth of the drawable (bits per pixel for the object).

display Specifies the connection to the X server.

root_return Returns the root window.

w Specifies the window whose current attributes you want to obtain.

width_return
height_return Return the drawable's dimensions (width and height).

window_attributes_return
Returns the specified window's attributes in the `XWindowAttributes` structure.

x_return
y_return Return the x and y coordinates that define the location of the

drawable. For a window, these coordinates specify the upper-left outer corner relative to its parent's origin. For pixmaps, these coordinates are always zero.

Description

The `XGetWindowAttributes` function returns the current attributes for the specified window to an `XWindowAttributes` structure.

`XGetWindowAttributes` can generate `BadDrawable` and `BadWindow` errors.

The `XGetGeometry` function returns the root window and the current geometry of the drawable. The geometry of the drawable includes the x and y coordinates, width and height, border width, and depth. These are described in the argument list. It is legal to pass to this function a window whose class is `InputOnly`.

Diagnostics

`BadDrawable`
: A value for a Drawable argument does not name a defined Window or Pixmap.

`BadWindow`
: A value for a Window argument does not name a defined Window.

See Also

XQueryPointer(3X11), XQueryTree(3X11)
Guide to the Xlib Library

Name

XGetWindowProperty, XListProperties, XChangeProperty, XRotateWindowProperties, XDeleteProperty – obtain and change window properties

Syntax

int XGetWindowProperty(*display*, *w*, *property*, *long_offset*, *long_length*, *delete*, *req_type*, *actual_type_return*, *actual_format_return*, *nitems_return*, *bytes_after_return*, *prop_return*)
Display **display*;
Window *w*;
Atom *property*;
long *long_offset*, *long_length*;
Bool *delete*;
Atom *req_type*;
Atom **actual_type_return*;
int **actual_format_return*;
unsigned long **nitems_return*;
unsigned long **bytes_after_return*;
unsigned char ***prop_return*;

Atom *XListProperties(*display*, *w*, *num_prop_return*)
Display **display*;
Window *w*;
int **num_prop_return*;

XChangeProperty(*display*, *w*, *property*, *type*, *format*, *mode*, *data*, *nelements*)
Display **display*;
Window *w*;
Atom *property*, *type*;
int *format*;
int *mode*;
unsigned char **data*;
int *nelements*;

XRotateWindowProperties(*display*, *w*, *properties*, *num_prop*, *npositions*)
Display **display*;
Window *w*;
Atom *properties*[];
int *num_prop*;
int *npositions*;

XDeleteProperty(*display*, *w*, *property*)
Display **display*;
Window *w*;
Atom *property*;

Arguments

actual_format_return
Returns the actual format of the property.

actual_type_return
Returns the atom identifier that defines the actual type of the property.

bytes_after_return
Returns the number of bytes remaining to be read in the property if a partial read was performed.

data Specifies the property data.

delete Specifies a Boolean value that determines whether the property is deleted.

display Specifies the connection to the X server.

format Specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities. Possible values are 8, 16, and 32. This information allows the X server to correctly perform byte-swap operations as necessary. If the format is 16-bit or 32-bit, you must explicitly cast your data pointer to a (char *) in the call to `XChangeProperty`.

long_length Specifies the length in 32-bit multiples of the data to be retrieved.

long_offset Specifies the offset in the specified property (in 32-bit quantities) where the data is to be retrieved.

mode Specifies the mode of the operation. You can pass `PropModeReplace`, `PropModePrepend`, or `PropModeAppend`.

nelements Specifies the number of elements of the specified data format.

nitems_return Returns the actual number of 8-bit, 16-bit, or 32-bit items stored in the prop_return data.

num_prop Specifies the length of the properties array.

num_prop_return
: Returns the length of the properties array.

npositions
: Specifies the rotation amount.

prop_return
: Returns a pointer to the data in the specified format.

property
: Specifies the property name.

properties
: Specifies the array of properties that are to be rotated.

req_type
: Specifies the atom identifier associated with the property type or `AnyPropertyType`.

type
: Specifies the type of the property. The X server does not interpret the type but simply passes it back to an application that later calls `XGetWindowProperty`.

w
: Specifies the window whose property you want to obtain, change, rotate or delete.

Description

The `XGetWindowProperty` function returns the actual type of the property; the actual format of the property; the number of 8-bit, 16-bit, or 32-bit items transferred; the number of bytes remaining to be read in the property; and a pointer to the data actually returned.
`XGetWindowProperty` sets the return arguments as follows:

- If the specified property does not exist for the specified window, `XGetWindowProperty` returns `None` to actual_type_return and the value zero to actual_format_return and bytes_after_return. The nitems_return argument is empty. In this case, the delete argument is ignored.
- If the specified property exists but its type does not match the specified type, `XGetWindowProperty` returns the actual property type to actual_type_return, the actual property format (never zero) to actual_format_return, and the property length in bytes (even if the actual_format_return is 16 or 32) to bytes_after_return. It also ignores the delete argument. The nitems_return argument is empty.
- If the specified property exists and either you assign `AnyPropertyType` to the req_type argument or the specified type matches the actual property type, `XGetWindowProperty` returns the actual property type to actual_type_return and the actual property format (never zero) to actual_format_return. It also returns a value to bytes_after_return and nitems_return, by defining the following values:

N = actual length of the stored property in bytes
(even if the format is 16 or 32)
I = 4 * long_offset
T = N - I
L = MINIMUM(T, 4 * long_length)
A = N - (I + L)

The returned value starts at byte index I in the property (indexing from zero), and its length in bytes is L. If the value for long_offset causes L to be negative, a `BadValue` error results. The value of bytes_after_return is A, giving the number of trailing unread bytes in the stored property.

`XGetWindowProperty` always allocates one extra byte in prop_return (even if the property is zero length) and sets it to ASCII null so that simple properties consisting of characters do not have to be copied into yet another string before use. If delete is `True` and bytes_after_return is zero, `XGetWindowProperty` deletes the property from the window and generates a `PropertyNotify` event on the window.

The function returns `Success` if it executes successfully. To free the resulting data, use `XFree`.

`XGetWindowProperty` can generate `BadAtom`, `BadValue`, and `BadWindow` errors.

The `XListProperties` function returns a pointer to an array of atom properties that are defined for the specified window or returns NULL if no properties were found. To free the memory allocated by this function, use `XFree`.

`XListProperties` can generate a `BadWindow` error.

The `XChangeProperty` function alters the property for the specified window and causes the X server to generate a `PropertyNotify` event on that window. `XChangeProperty` performs the following:

- If mode is `PropModeReplace`, `XChangeProperty` discards the previous property value and stores the new data.
- If mode is `PropModePrepend` or `PropModeAppend`, `XChangeProperty` inserts the specified data before the beginning of the existing data or onto the end of the existing data, respectively. The type and format must match the existing property value, or a `BadMatch` error results. If the property is undefined, it is treated as defined with the correct type and format with zero-length data.

The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, until the window is destroyed, or until the server resets. For a discussion of what happens when the connection to the X server is closed, see section 2.5. The maximum size of a property is server dependent and can vary dynamically depending on the amount of memory the server has available. (If there is insufficient space, a `BadAlloc` error results.)

`XChangeProperty` can generate `BadAlloc`, `BadAtom`, `BadMatch`, `BadValue`, and `BadWindow` errors.

The `XRotateWindowProperties` function allows you to rotate properties on a window and causes the X server to generate `PropertyNotify` events. If the property names in the properties array are viewed as being numbered starting from zero and if there are num_prop property names in the list, then the value associated with property name I becomes the value associated with property name (I + npositions) mod N for all I from zero to N – 1. The effect is to rotate the states by npositions places around the virtual ring of property names (right for positive npositions, left for negative npositions). If npositions mod N is nonzero, the X server generates a `PropertyNotify` event for each property in the order that they are listed in the array. If an atom occurs more than once in the list or no property with that name is defined for the window, a `BadMatch` error results. If a `BadAtom` or `BadMatch` error results, no properties are changed.

`XRotateWindowProperties` can generate `BadAtom`, `BadMatch`, and `BadWindow` errors.

The `XDeleteProperty` function deletes the specified property only if the property was defined on the specified window and causes the X server to generate a `PropertyNotify` event on the window unless the property does not exist.

`XDeleteProperty` can generate `BadAtom` and `BadWindow` errors.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadAtom`	A value for an Atom argument does not name a defined Atom.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified

for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

XInternAtom(3X11)
Guide to the Xlib Library

Name

XGrabButton, XUngrabButton – grab pointer buttons

Syntax

XGrabButton(*display*, *button*, *modifiers*, *grab_window*, *owner_events*, *event_mask*, *pointer_mode*, *keyboard_mode*, *confine_to*, *cursor*)
Display **display*;
unsigned int *button*;
unsigned int *modifiers*;
Window *grab_window*;
Bool *owner_events*;
unsigned int *event_mask*;
int *pointer_mode*, *keyboard_mode*;
Window *confine_to*;
Cursor *cursor*;

XUngrabButton(*display*, *button*, *modifiers*, *grab_window*)
Display **display*;
unsigned int *button*;
unsigned int *modifiers*;
Window *grab_window*;

Arguments

button	Specifies the pointer button that is to be grabbed or released or `AnyButton`.
confine_to	Specifies the window to confine the pointer in or `None`.
cursor	Specifies the cursor that is to be displayed or `None`.
display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass `GrabModeSync` or `GrabModeAsync`.
modifiers	Specifies the set of keymasks or `AnyModifier`. The mask is the bitwise inclusive OR of the valid keymask bits.

owner_events Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.

pointer_mode Specifies further processing of pointer events. You can pass `GrabModeSync` or `GrabModeAsync`.

Description

The `XGrabButton` function establishes a passive grab. In the future, the pointer is actively grabbed (as for `XGrabPointer`), the last-pointer-grab time is set to the time at which the button was pressed (as transmitted in the `ButtonPress` event), and the `ButtonPress` event is reported if all of the following conditions are true:

- The pointer is not grabbed, and the specified button is logically pressed when the specified modifier keys are logically down, and no other buttons or modifier keys are logically down.
- The grab_window contains the pointer.
- The confine_to window (if any) is viewable.
- A passive grab on the same button/key combination does not exist on any ancestor of grab_window.

The interpretation of the remaining arguments is as for `XGrabPointer`. The active grab is terminated automatically when the logical state of the pointer has all buttons released (independent of the state of the logical modifier keys).

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

This request overrides all previous grabs by the same client on the same button/key combinations on the same window. A modifiers of `AnyModifier` is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned KeyCodes. A button of `AnyButton` is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the specified button currently be assigned to a physical button.

If some other client has already issued a `XGrabButton` with the same button/key combination on the same window, a `BadAccess` error results. When using `AnyModifier` or `AnyButton`, the request fails completely, and a `BadAccess` error results (no grabs are established) if there is a conflicting grab for any combination. `XGrabButton` has no effect on an

active grab.

XGrabButton can generate BadCursor, BadValue, and BadWindow errors.

The XUngrabButton function releases the passive button/key combination on the specified window if it was grabbed by this client. A modifiers of AnyModifier is equivalent to issuing the ungrab request for all possible modifier combinations, including the combination of no modifiers. A button of AnyButton is equivalent to issuing the request for all possible buttons. XUngrabButton has no effect on an active grab.

XUngrabButton can generate BadValue and BadWindow errors.

Diagnostics

BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XAllowEvents(3X11), XGrabPointer(3X11), XGrabKey(3X11), XGrabKeyboard(3X11),
Guide to the Xlib Library

XGrabKey (3X11)

Name

XGrabKey, XUngrabKey – grab keyboard keys

Syntax

XGrabKey(*display*, *keycode*, *modifiers*, *grab_window*, *owner_events*, *pointer_mode*, *keyboard_mode*)
Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *grab_window*;
Bool *owner_events*;
int *pointer_mode*, *keyboard_mode*;

XUngrabKey(*display*, *keycode*, *modifiers*, *grab_window*)
Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *grab_window*;

Arguments

display	Specifies the connection to the X server.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass `GrabModeSync` or `GrabModeAsync`.
keycode	Specifies the KeyCode or `AnyKey`.
modifiers	Specifies the set of keymasks or `AnyModifier`. The mask is the bitwise inclusive OR of the valid keymask bits.
owner_events	Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
pointer_mode	Specifies further processing of pointer events. You can pass `GrabModeSync` or `GrabModeAsync`.

Description

The XGrabKey function establishes a passive grab on the keyboard. In the future, the keyboard is actively grabbed (as for XGrabKeyboard), the last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the KeyPress event), and the KeyPress event is reported if all of the following conditions are true:

- The keyboard is not grabbed and the specified key (which can itself be a modifier key) is logically pressed when the specified modifier keys are logically down, and no other modifier keys are logically down.
- Either the grab_window is an ancestor of (or is) the focus window, or the grab_window is a descendant of the focus window and contains the pointer.
- A passive grab on the same key combination does not exist on any ancestor of grab_window.

The interpretation of the remaining arguments is as for XGrabKeyboard. The active grab is terminated automatically when the logical state of the keyboard has the specified key released (independent of the logical state of the modifier keys).

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

A modifiers argument of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned KeyCodes. A keycode argument of AnyKey is equivalent to issuing the request for all possible KeyCodes. Otherwise, the specified keycode must be in the range specified by min_keycode and max_keycode in the connection setup, or a BadValue error results.

If some other client has issued a XGrabKey with the same key combination on the same window, a BadAccess error results. When using AnyModifier or AnyKey, the request fails completely, and a BadAccess error results (no grabs are established) if there is a conflicting grab for any combination.

XGrabKey can generate BadAccess, BadValue, and BadWindow errors.

The XUngrabKey function releases the key combination on the specified window if it was grabbed by this client. It has no effect on an active grab. A modifiers of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers).

A keycode argument of `AnyKey` is equivalent to issuing the request for all possible key codes.

`XUngrabKey` can generate `BadValue` and `BadWindow` error.

Diagnostics

`BadAccess`	A client attempted to grab a key/button combination already grabbed by another client.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XAllowAccess(3X11), XGrabButton(3X11), XGrabKeyboard(3X11), XGrabPointer(3X11)
Guide to the Xlib Library

Name

XGrabKeyboard, XUngrabKeyboard – grab the keyboard

Syntax

int XGrabKeyboard(*display*, *grab_window*, *owner_events*, *pointer_mode*, *keyboard_mode*, *time*)
Display **display*;
Window *grab_window*;
Bool *owner_events*;
int *pointer_mode*, *keyboard_mode*;
Time *time*;

XUngrabKeyboard(*display*, *time*)
Display **display*;
Time *time*;

Arguments

display	Specifies the connection to the X server.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass `GrabModeSync` or `GrabModeAsync`.
owner_events	Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
pointer_mode	Specifies further processing of pointer events. You can pass `GrabModeSync` or `GrabModeAsync`.
time	Specifies the time. You can pass either a timestamp or `CurrentTime`.

Description

The `XGrabKeyboard` function actively grabs control of the keyboard and generates `FocusIn` and `FocusOut` events. Further key events are reported only to the grabbing client. `XGrabKeyboard` overrides any active keyboard grab by this client. If owner_events is `False`, all generated key events are reported with respect to grab_window. If owner_events is `True` and if a generated key event would normally be reported to this client, it is reported normally; otherwise, the event is reported with respect to the

grab_window. Both KeyPress and KeyRelease events are always reported, independent of any event selection made by the client.

If the keyboard_mode argument is GrabModeAsync, keyboard event processing continues as usual. If the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If the keyboard_mode argument is GrabModeSync, the state of the keyboard (as seen by client applications) appears to freeze, and the X server generates no further keyboard events until the grabbing client issues a releasing XAllowEvents call or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued in the server for later processing.

If pointer_mode is GrabModeAsync, pointer event processing is unaffected by activation of the grab. If pointer_mode is GrabModeSync, the state of the pointer (as seen by client applications) appears to freeze, and the X server generates no further pointer events until the grabbing client issues a releasing XAllowEvents call or until the keyboard grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If the keyboard is actively grabbed by some other client, XGrabKeyboard fails and returns AlreadyGrabbed. If grab_window is not viewable, it fails and returns GrabNotViewable. If the keyboard is frozen by an active grab of another client, it fails and returns GrabFrozen. If the specified time is earlier than the last-keyboard-grab time or later than the current X server time, it fails and returns GrabInvalidTime. Otherwise, the last-keyboard-grab time is set to the specified time (CurrentTime is replaced by the current X server time).

XGrabKeyboard can generate BadValue and BadWindow errors.

The XUngrabKeyboard function releases the keyboard and any queued events if this client has it actively grabbed from either XGrabKeyboard or XGrabKey. XUngrabKeyboard does not release the keyboard and any queued events if the specified time is earlier than the last-keyboard-grab time or is later than the current X server time. It also generates FocusIn and FocusOut events. The X server automatically performs an UngrabKeyboard request if the event window for an active keyboard grab becomes not viewable.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XAllowEvents(3X11), XGrabButton(3X11), XGrabKey(3X11), XGrabPointer(3X11)
Guide to the Xlib Library

XGrabPointer (3X11)

Name

XGrabPointer, XUngrabPointer, XChangeActivePointerGrab – grab the pointer

Syntax

int XGrabPointer(*display*, *grab_window*, *owner_events*, *event_mask*, *pointer_mode*, *keyboard_mode*, *confine_to*, *cursor*, *time*)
Display **display*;
Window *grab_window*;
Bool *owner_events*;
unsigned int *event_mask*;
int *pointer_mode*, *keyboard_mode*;
Window *confine_to*;
Cursor *cursor*;
Time *time*;

XUngrabPointer(*display*, *time*)
Display **display*;
Time *time*;

XChangeActivePointerGrab(*display*, *event_mask*, *cursor*, *time*)
Display **display*;
unsigned int *event_mask*;
Cursor *cursor*;
Time *time*;

Arguments

confine_to	Specifies the window to confine the pointer in or `None`.
cursor	Specifies the cursor that is to be displayed during the grab or `None`.
display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass `GrabModeSync` or `GrabModeAsync`.

owner_events	Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
pointer_mode	Specifies further processing of pointer events. You can pass `GrabModeSync` or `GrabModeAsync`.
time	Specifies the time. You can pass either a timestamp or `CurrentTime`.

Description

The `XGrabPointer` function actively grabs control of the pointer and returns `GrabSuccess` if the grab was successful. Further pointer events are reported only to the grabbing client. `XGrabPointer` overrides any active pointer grab by this client. If owner_events is `False`, all generated pointer events are reported with respect to grab_window and are reported only if selected by event_mask. If owner_events is `True` and if a generated pointer event would normally be reported to this client, it is reported as usual. Otherwise, the event is reported with respect to the grab_window and is reported only if selected by event_mask. For either value of owner_events, unreported events are discarded.

If the pointer_mode is `GrabModeAsync`, pointer event processing continues as usual. If the pointer is currently frozen by this client, the processing of events for the pointer is resumed. If the pointer_mode is `GrabModeSync`, the state of the pointer, as seen by client applications, appears to freeze, and the X server generates no further pointer events until the grabbing client calls `XAllowEvents` or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If the keyboard_mode is `GrabModeAsync`, keyboard event processing is unaffected by activation of the grab. If the keyboard_mode is `GrabModeSync`, the state of the keyboard, as seen by client applications, appears to freeze, and the X server generates no further keyboard events until the grabbing client calls `XAllowEvents` or until the pointer grab is released. Actual keyboard changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If a cursor is specified, it is displayed regardless of what window the pointer is in. If `None` is specified, the normal cursor for that window is displayed when the pointer is in grab_window or one of its subwindows; otherwise, the cursor for grab_window is displayed.

If a confine_to window is specified, the pointer is restricted to stay contained in that window. The confine_to window need have no relationship to the grab_window. If the pointer is not initially in the confine_to window, it is warped automatically to the closest edge just before the grab activates and enter/leave events are generated as usual. If the confine_to window is subsequently reconfigured, the pointer is warped automatically, as necessary, to keep it contained in the window.

The time argument allows you to avoid certain circumstances that come up if applications take a long time to respond or if there are long network delays. Consider a situation where you have two applications, both of which normally grab the pointer when clicked on. If both applications specify the timestamp from the event, the second application may wake up faster and successfully grab the pointer before the first application. The first application then will get an indication that the other application grabbed the pointer before its request was processed.

`XGrabPointer` generates `EnterNotify` and `LeaveNotify` events.

Either if grab_window or confine_to window is not viewable or if the confine_to window lies completely outside the boundaries of the root window, `XGrabPointer` fails and returns `GrabNotViewable`. If the pointer is actively grabbed by some other client, it fails and returns `AlreadyGrabbed`. If the pointer is frozen by an active grab of another client, it fails and returns `GrabFrozen`. If the specified time is earlier than the last-pointer-grab time or later than the current X server time, it fails and returns `GrabInvalidTime`. Otherwise, the last-pointer-grab time is set to the specified time (`CurrentTime` is replaced by the current X server time).

`XGrabPointer` can generate `BadCursor`, `BadValue`, and `BadWindow` errors.

The `XUngrabPointer` function releases the pointer and any queued events if this client has actively grabbed the pointer from `XGrabPointer`, `XGrabButton`, or from a normal button press. `XUngrabPointer` does not release the pointer if the specified time is earlier than the last-pointer-grab time or is later than the current X server time. It also generates `EnterNotify` and `LeaveNotify` events. The X server performs an `UngrabPointer` request automatically if the event window or confine_to window for an active pointer grab becomes not viewable or if window reconfiguration causes the confine_to window to lie completely outside the boundaries of the root window.

The `XChangeActivePointerGrab` function changes the specified dynamic parameters if the pointer is actively grabbed by the client and if the specified time is no earlier than the last-pointer-grab time and no later than

the current X server time. This function has no effect on the passive parameters of a XGrabButton. The interpretation of event_mask and cursor is the same as described in XGrabPointer.

XChangeActivePointerGrab can generate a BadCursor and BadValue error.

Diagnostics

BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XAllowEvents(3X11), XGrabButton(3X11), XGrabKey(3X11), XGrabKeyboard(3X11)
Guide to the Xlib Library

XGrabServer (3X11)

Name

XGrabServer, XUngrabServer – grab the server

Syntax

XGrabServer(*display*)
    Display **display*;

XUngrabServer(*display*)
    Display **display*;

Arguments

display	Specifies the connection to the X server.

Description

The `XGrabServer` function disables processing of requests and close downs on all other connections than the one this request arrived on. You should not grab the X server any more than is absolutely necessary.

The `XUngrabServer` function restarts processing of requests and close downs on other connections. You should avoid grabbing the X server as much as possible.

See Also

XGrabButton(3X11), XGrabKey(3X11), XGrabKeyboard(3X11), XGrabPointer(3X11)
Guide to the Xlib Library

Name

XIfEvent, XCheckIfEvent, XPeekIfEvent – check the event queue with a predicate procedure

Syntax

XIfEvent(*display*, *event_return*, *predicate*, *arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

Bool XCheckIfEvent(*display*, *event_return*, *predicate*, *arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

XPeekIfEvent(*display*, *event_return*, *predicate*, *arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

Arguments

arg	Specifies the user-supplied argument that will be passed to the predicate procedure.
display	Specifies the connection to the X server.
event_return	Returns either a copy of or the matched event's associated structure.
predicate	Specifies the procedure that is to be called to determine if the next event in the queue matches what you want.

Description

The `XIfEvent` function completes only when the specified predicate procedure returns `True` for an event, which indicates an event in the queue matches. `XIfEvent` flushes the output buffer if it blocks waiting for additional events. `XIfEvent` removes the matching event from the queue and copies the structure into the client-supplied `XEvent` structure.

When the predicate procedure finds a match, `XCheckIfEvent` copies the matched event into the client-supplied `XEvent` structure and returns `True`. (This event is removed from the queue.) If the predicate procedure finds no match, `XCheckIfEvent` returns `False`, and the output buffer will have been flushed. All earlier events stored in the queue are not discarded.

The `XPeekIfEvent` function returns only when the specified predicate procedure returns `True` for an event. After the predicate procedure finds a match, `XPeekIfEvent` copies the matched event into the client-supplied `XEvent` structure without removing the event from the queue. `XPeekIfEvent` flushes the output buffer if it blocks waiting for additional events.

See Also

XPutBackEvent(3X11) XNextEvent(3X11), XSendEvent(3X11)
Guide to the Xlib Library

Name

XInstallColormap, XUninstallColormap, XListInstalledColormaps – control colormaps

Syntax

XInstallColormap(*display*, *colormap*)
Display **display*;
Colormap *colormap*;

XUninstallColormap(*display*, *colormap*)
Display **display*;
Colormap *colormap*;

Colormap *XListInstalledColormaps(*display*, *w*, *num_return*)
Display **display*;
Window *w*;
int **num_return*;

Arguments

colormap	Specifies the colormap.
display	Specifies the connection to the X server.
num_return	Returns the number of currently installed colormaps.
w	Specifies the window that determines the screen.

Description

The `XInstallColormap` function installs the specified colormap for its associated screen. All windows associated with this colormap immediately display with true colors. You associated the windows with this colormap when you created them by calling `XCreateWindow`, `XCreateSimpleWindow`, `XChangeWindowAttributes`, or `XSetWindowColormap`.

If the specified colormap is not already an installed colormap, the X server generates a `ColormapNotify` event on each window that has that colormap. In addition, for every other colormap that is installed as a result of a call to `XInstallColormap`, the X server generates a `ColormapNotify` event on each window that has that colormap.

XInstallColormap can generate a BadColor error.

The XUninstallColormap function removes the specified colormap from the required list for its screen. As a result, the specified colormap might be uninstalled, and the X server might implicitly install or uninstall additional colormaps. Which colormaps get installed or uninstalled is server-dependent except that the required list must remain installed.

If the specified colormap becomes uninstalled, the X server generates a ColormapNotify event on each window that has that colormap. In addition, for every other colormap that is installed or uninstalled as a result of a call to XUninstallColormap, the X server generates a ColormapNotify event on each window that has that colormap.

XUninstallColormap can generate a BadColor error.

The XListInstalledColormaps function returns a list of the currently installed colormaps for the screen of the specified window. The order of the colormaps in the list is not significant and is no explicit indication of the required list. When the allocated list is no longer needed, free it by using XFree.

XListInstalledColormaps can generate a BadWindow error.

Diagnostics

BadColor	A value for a Colormap argument does not name a defined Colormap.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

Guide to the Xlib Library

Name

XInternAtom, XGetAtomName – create or return atom names

Syntax

Atom XInternAtom(*display*, *atom_name*, *only_if_exists*)
    Display **display*;
    char **atom_name*;
    Bool *only_if_exists*;

char *XGetAtomName(*display*, *atom*)
    Display **display*;
    Atom *atom*;

Arguments

atom	Specifies the atom for the property name you want returned.
atom_name	Specifies the name associated with the atom you want returned.
display	Specifies the connection to the X server.
only_if_exists	Specifies a Boolean value that indicates whether `XInternAtom` creates the atom.

Description

The `XInternAtom` function returns the atom identifier associated with the specified atom_name string. If only_if_exists is `False`, the atom is created if it does not exist. Therefore, `XInternAtom` can return `None`. You should use a null-terminated ISO Latin-1 string for atom_name. Case matters; the strings *thing*, *Thing*, and *thinG* all designate different atoms. The atom will remain defined even after the client's connection closes. It will become undefined only when the last connection to the X server closes.

`XInternAtom` can generate `BadAlloc` and `BadValue` errors.

The `XGetAtomName` function returns the name associated with the specified atom. To free the resulting string, call `XFree`.

`XGetAtomName` can generate a `BadAtom` error.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadAtom`	A value for an Atom argument does not name a defined Atom.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XGetWindowProperty(3X11)
Guide to the Xlib Library

Name

XIntersectRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XXorRegion, XOffsetRegion, XShrinkRegion – region arithmetic utilities

Syntax

XIntersectRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XUnionRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XUnionRectWithRegion(*rectangle*, *src_region*, *dest_region_return*)
XRectangle **rectangle*;
Region *src_region*;
Region *dest_region_return*;

XSubtractRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XXorRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XOffsetRegion(*r*, *dx*, *dy*)
Region *r*;
int *dx*, *dy*;

XShrinkRegion(*r*, *dx*, *dy*)
Region *r*;
int *dx*, *dy*;

Arguments

dest_region_return
Returns the destination region.

dr_return Returns the result of the computation. ds Dy move or shrink

dx

dy Specify the x and y coordinates, which define the amount you want to the specified region.

r Specifies the region.

rectangle Specifies the rectangle.

sra

srb Specify the two regions with which you want to perform the computation.

src_region Specifies the source region to be used.

Description

The `XIntersectRegion` function computes the intersection of two regions.

The `XUnionRegion` function computes the union of two regions.

The `XUnionRectWithRegion` function updates the destination region from a union of the specified rectangle and the specified source region.

The `XSubtractRegion` function subtracts srb from sra and stores the results in dr_return.

The `XXorRegion` function calculates the difference between the union and intersection of two regions.

The `XOffsetRegion` function moves the specified region by a specified amount.

The `XShrinkRegion` function reduces the specified region by a specified amount. Positive values shrink the size of the region, and negative values expand the region.

See Also

XCreateRegion(3X11), XEmptyRegion(3X11),
Guide to the Xlib Library

Name

XListFonts, XFreeFontNames, XListFontsWithInfo, XFreeFontInfo – obtain or free font names and information

Syntax

char **XListFonts(*display*, *pattern*, *maxnames*, *actual_count_return*)
    Display **display*;
    char **pattern*;
    int *maxnames*;
    int **actual_count_return*;

XFreeFontNames(*list*)
    char **list*[];

char **XListFontsWithInfo(*display*, *pattern*, *maxnames*, *count_return*, *info_return*)
    Display **display*;
    char **pattern*;
    int *maxnames*;
    int **count_return*;
    XFontStruct ***info_return*;

XFreeFontInfo(*names*, *free_info*, *actual_count*)
    char ***names*;
    XFontStruct **free_info*;
    int *actual_count*;

Arguments

actual_count	Specifies the actual number of matched font names returned by `XListFontsWithInfo`.
actual_count_return	Returns the actual number of font names.
count_return	Returns the actual number of matched font names.
display	Specifies the connection to the X server.
info_return	Returns a pointer to the font information.
free_info	Specifies the pointer to the font information returned by `XListFontsWithInfo`.
list	Specifies the array of strings you want to free.

maxnames	Specifies the maximum number of names to be returned.
names	Specifies the list of font names returned by `XListFontsWithInfo`.
pattern	Specifies the null-terminated pattern string that can contain wildcard characters.

Description

The `XListFonts` function returns an array of available font names (as controlled by the font search path; see `XSetFontPath`) that match the string you passed to the pattern argument. The string should be ISO Latin-1; uppercase and lowercase do not matter. Each string is terminated by an ASCII null. The pattern string can contain any characters, but each asterisk (*) is a wildcard for any number of characters, and each question mark (?) is a wildcard for a single character. The client should call `XFreeFontNames` when finished with the result to free the memory.

The `XFreeFontNames` function frees the array and strings returned by `XListFonts` or `XListFontsWithInfo`.

The `XListFontsWithInfo` function returns a list of font names that match the specified pattern and their associated font information. The list of names is limited to size specified by maxnames. The information returned for each font is identical to what `XLoadQueryFont` would return except that the per-character metrics are not returned. The pattern string can contain any characters, but each asterisk (*) is a wildcard for any number of characters, and each question mark (?) is a wildcard for a single character. To free the allocated name array, the client should call `XFreeFontNames`. To free the the font information array, the client should call `XFreeFontInfo`.

The `XFreeFontInfo` function frees the the font information array.

See Also

XLoadFont(3X11), XSetFontPath(3X11)
Guide to the Xlib Library

Name

XLoadFont, XQueryFont, XLoadQueryFont, XFreeFont, XGetFontProperty, XUnloadFont – load or unload fonts

Syntax

Font XLoadFont(*display*, *name*)
Display **display*;
char **name*;

XFontStruct *XQueryFont(*display*, *font_ID*)
Display **display*;
XID *font_ID*;

XFontStruct *XLoadQueryFont(*display*, *name*)
Display **display*;
char **name*;

XFreeFont(*display*, *font_struct*)
Display **display*;
XFontStruct **font_struct*;

Bool XGetFontProperty(*font_struct*, *atom*, *value_return*)
XFontStruct **font_struct*;
Atom *atom*;
unsigned long **value_return*;

XUnloadFont(*display*, *font*)
Display **display*;
Font *font*;

Arguments

atom	Specifies the atom for the property name you want returned.
display	Specifies the connection to the X server.
font	Specifies the font.
font_ID	Specifies the font ID or the `GContext` ID.
font_struct	Specifies the storage associated with the font.
gc	Specifies the GC.
name	Specifies the name of the font, which is a null-terminated string.
value_return	Returns the value of the font property.

Description

The XLoadFont function loads the specified font and returns its associated font ID. The name should be ISO Latin-1 encoding; uppercase and lowercase do not matter. If XLoadFont was unsuccessful at loading the specified font, a BadName error results. Fonts are not associated with a particular screen and can be stored as a component of any GC. When the font is no longer needed, call XUnloadFont.

XLoadFont can generate BadAlloc and BadName errors.

The XQueryFont function returns a pointer to the XFontStruct structure, which contains information associated with the font. You can query a font or the font stored in a GC. The font ID stored in the XFontStruct structure will be the GContext ID, and you need to be careful when using this ID in other functions (see XGContextFromGC). To free this data, use XFreeFontInfo.

XLoadQueryFont can generate a BadAlloc error.

The XLoadQueryFont function provides the most common way for accessing a font. XLoadQueryFont both opens (loads) the specified font and returns a pointer to the appropriate XFontStruct structure. If the font does not exist, XLoadQueryFont returns NULL.

The XFreeFont function deletes the association between the font resource ID and the specified font and frees the XFontStruct structure. The font itself will be freed when no other resource references it. The data and the font should not be referenced again.

XFreeFont can generate a BadFont error.

Given the atom for that property, the XGetFontProperty function returns the value of the specified font property. XGetFontProperty also returns False if the property was not defined or True if it was defined. A set of predefined atoms exists for font properties, which can be found in <X11/Xatom.h>. This set contains the standard properties associated with a font. Although it is not guaranteed, it is likely that the predefined font properties will be present.

The XUnloadFont function deletes the association between the font resource ID and the specified font. The font itself will be freed when no other resource references it. The font should not be referenced again.

XUnloadFont can generate a BadFont error.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadFont`	A value for a Font or GContext argument does not name a defined Font.
`BadName`	A font or color of the specified name does not exist.

See Also

XListFonts(3X11), XSetFontPath(3X11)
Guide to the Xlib Library

XLookupKeysym (3X11)

Name

XLookupKeysym, XRefreshKeyboardMapping, XLookupString, XRebindKeySym – handle keyboard input events

Syntax

KeySym XLookupKeysym(*key_event, index*)
    XKeyEvent **key_event*;
    int *index*;

XRefreshKeyboardMapping(*event_map*)
    XMappingEvent **event_map*;

int XLookupString(*event_struct, buffer_return, bytes_buffer, keysym_return, status_in_out*)
    XKeyEvent **event_struct*;
    char **buffer_return*;
    int *bytes_buffer*;
    KeySym **keysym_return*;
    XComposeStatus **status_in_out*;

XRebindKeysym(*display, keysym, list, mod_count, string, bytes_string*)
    Display **display*;
    KeySym *keysym*;
    KeySym *list*[];
    int *mod_count*;
    unsigned char **string*;
    int *bytes_string*;

Arguments

buffer_return	Returns the translated characters.
bytes_buffer	Specifies the length of the buffer. No more than bytes_buffer of translation are returned.
bytes_string	Specifies the length of the string.
display	Specifies the connection to the X server.
event_map	Specifies the mapping event that is to be used.
event_struct	Specifies the key event structure to be used. You can pass `XKeyPressedEvent` or `XKeyReleasedEvent`.
index	Specifies the index into the KeySyms list for the event's KeyCode.

key_event	Specifies the `KeyPress` or `KeyRelease` event.
keysym	Specifies the KeySym that is to be tested.
keysym_return	Returns the KeySym computed from the event if this argument is not NULL.
list	Specifies the KeySyms to be used as modifiers.
mod_count	Specifies the number of modifiers in the modifier list.
status_in_out	Specifies or returns the `XComposeStatus` structure or NULL.
string	Specifies a pointer to the string that is copied and will be returned by `XLookupString`.

Description

The `XLookupKeysym` function uses a given keyboard event and the index you specified to return the KeySym from the list that corresponds to the KeyCode member in the `XKeyPressedEvent` or `XKeyReleasedEvent` structure. If no KeySym is defined for the KeyCode of the event, `XLookupKeysym` returns `NoSymbol`.

The `XRefreshKeyboardMapping` function refreshes the stored modifier and keymap information. You usually call this function when a `MappingNotify` event with a request member of `MappingKeyboard` or `MappingModifier` occurs. The result is to update Xlib's knowledge of the keyboard.

The `XLookupString` function is a convenience routine that maps a key event to an ISO Latin-1 string, using the modifier bits in the key event to deal with shift, lock, and control. It returns the translated string into the user's buffer. It also detects any rebound KeySyms (see `XRebindKeysym`) and returns the specified bytes. `XLookupString` returns the length of the string stored in the tag buffer. If the lock modifier has the caps lock KeySym associated with it, `XLookupString` interprets the lock modifier to perform caps lock processing.

If present (non-NULL), the `XComposeStatus` structure records the state, which is private to Xlib, that needs preservation across calls to `XLookupString` to implement compose processing.

The `XRebindKeysym` function can be used to rebind the meaning of a KeySym for the client. It does not redefine any key in the X server but merely provides an easy way for long strings to be attached to keys. `XLookupString` returns this string when the appropriate set of modifier keys are pressed and when the KeySym would have been used for the

translation. Note that you can rebind a KeySym that may not exist.

See Also

XStringToKeysym(3X11)
Guide to the Xlib Library

Name

XMapWindow, XMapRaised, XMapSubwindows – map windows

Syntax

XMapWindow(*display*, *w*)
Display **display*;
Window *w*;

XMapRaised(*display*, *w*)
Display **display*;
Window *w*;

XMapSubwindows(*display*, *w*)
Display **display*;
Window *w*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.

Description

The `XMapWindow` function maps the window and all of its subwindows that have had map requests. Mapping a window that has an unmapped ancestor does not display the window but marks it as eligible for display when the ancestor becomes mapped. Such a window is called unviewable. When all its ancestors are mapped, the window becomes viewable and will be visible on the screen if it is not obscured by another window. This function has no effect if the window is already mapped.

If the override-redirect of the window is `False` and if some other client has selected `SubstructureRedirectMask` on the parent window, then the X server generates a `MapRequest` event, and the `XMapWindow` function does not map the window. Otherwise, the window is mapped, and the X server generates a `MapNotify` event.

If the window becomes viewable and no earlier contents for it are remembered, the X server tiles the window with its background. If the window's background is undefined, the existing screen contents are not altered, and the X server generates zero or more `Expose` events. If backing-store was maintained while the window was unmapped, no `Expose` events are generated. If backing-store will now be maintained, a full-window

exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

If the window is an `InputOutput` window, `XMapWindow` generates `Expose` events on each `InputOutput` window that it causes to be displayed. If the client maps and paints the window and if the client begins processing events, the window is painted twice. To avoid this, first ask for `Expose` events and then map the window, so the client processes input events as usual. The event list will include `Expose` for each window that has appeared on the screen. The client's normal response to an `Expose` event should be to repaint the window. This method usually leads to simpler programs and to proper interaction with window managers.

`XMapWindow` can generate a `BadWindow` error.

The `XMapRaised` function essentially is similar to `XMapWindow` in that it maps the window and all of its subwindows that have had map requests. However, it also raises the specified window to the top of the stack.

`XMapRaised` can generate a `BadWindow` error.

The `XMapSubwindows` function maps all subwindows for a specified window in top-to-bottom stacking order. The X server generates `Expose` events on each newly displayed window. This may be much more efficient than mapping many windows one at a time because the server needs to perform much of the work only once, for all of the windows, rather than for each window.

`XMapSubwindows` can generate a `BadWindow` error.

Diagnostics

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

XChangeWindowAttributes(3X11), XConfigureWindow(3X11), XCreateWindow(3X11), XDestroyWindow(3X11), XRaiseWindow(3X11), XUnmapWindow(3X11)
Guide to the Xlib Library

Name

NextEvent, XPeekEvent, XWindowEvent, XCheckWindowEvent, XMaskEvent, XCheckMaskEvent, XCheckTypedEvent, XCheckTypedWindowEvent – select events by type

Syntax

XNextEvent(*display*, *event_return*)
Display **display*;
XEvent **event_return*;

XPeekEvent(*display*, *event_return*)
Display **display*;
XEvent **event_return*;

XWindowEvent(*display*, *w*, *event_mask*, *event_return*)
Display **display*;
Window *w*;
long *event_mask*;
XEvent **event_return*;

Bool XCheckWindowEvent(*display*, *w*, *event_mask*, *event_return*)
Display **display*;
Window *w*;
long *event_mask*;
XEvent **event_return*;

XMaskEvent(*display*, *event_mask*, *event_return*)
Display **display*;
long *event_mask*;
XEvent **event_return*;

Bool XCheckMaskEvent(*display*, *event_mask*, *event_return*)
Display **display*;
long *event_mask*;
XEvent **event_return*;

Bool XCheckTypedEvent(*display*, *event_type*, *event_return*)
Display **display*;
int *event_type*;
XEvent **event_return*;

Bool XCheckTypedWindowEvent(*display*, *w*, *event_type*, *event_return*)
Display **display*;
Window *w*;

int *event_type*;
XEvent **event_return*;

Arguments

display	Specifies the connection to the X server.
event_mask	Specifies the event mask.
event_return	Returns the matched event's associated structure.
event_return	Returns the next event in the queue.
event_return	Returns a copy of the matched event's associated structure.
event_type	Specifies the event type to be compared.
w	Specifies the window whose event uou are interested in.

Description

The `XNextEvent` function copies the first event from the event queue into the specified `XEvent` structure and then removes it from the queue. If the event queue is empty, `XNextEvent` flushes the output buffer and blocks until an event is received.

The `XPeekEvent` function returns the first event from the event queue, but it does not remove the event from the queue. If the queue is empty, `XPeekEvent` flushes the output buffer and blocks until an event is received. It then copies the event into the client-supplied `XEvent` structure without removing it from the event queue.

The `XWindowEvent` function searches the event queue for an event that matches both the specified window and event mask. When it finds a match, `XWindowEvent` removes that event from the queue and copies it into the specified `XEvent` structure. The other events stored in the queue are not discarded. If a matching event is not in the queue, `XWindowEvent` flushes the output buffer and blocks until one is received.

The `XCheckWindowEvent` function searches the event queue and then the events available on the server connection for the first event that matches the specified window and event mask. If it finds a match, `XCheckWindowEvent` removes that event, copies it into the specified `XEvent` structure, and returns `True`. The other events stored in the queue are not discarded. If the event you requested is not available, `XCheckWindowEvent` returns `False`, and the output buffer will have been flushed.

The XMaskEvent function searches the event queue for the events associated with the specified mask. When it finds a match, XMaskEvent removes that event and copies it into the specified XEvent structure. The other events stored in the queue are not discarded. If the event you requested is not in the queue, XMaskEvent flushes the output buffer and blocks until one is received.

The XCheckMaskEvent function searches the event queue and then any events available on the server connection for the first event that matches the specified mask. If it finds a match, XCheckMaskEvent removes that event, copies it into the specified XEvent structure, and returns True. The other events stored in the queue are not discarded. If the event you requested is not available, XCheckMaskEvent returns False, and the output buffer will have been flushed.

The XCheckTypedEvent function searches the event queue and then any events available on the server connection for the first event that matches the specified type. If it finds a match, XCheckTypedEvent removes that event, copies it into the specified XEvent structure, and returns True. The other events in the queue are not discarded. If the event is not available, XCheckTypedEvent returns False, and the output buffer will have been flushed.

The XCheckTypedWindowEvent function searches the event queue and then any events available on the server connection for the first event that matches the specified type and window. If it finds a match, XCheckTypedWindowEvent removes the event from the queue, copies it into the specified XEvent structure, and returns True. The other events in the queue are not discarded. If the event is not available, XCheckTypedWindowEvent returns False, and the output buffer will have been flushed.

See Also

XIfEvent(3X11), XPutBackEvent(3X11), XSendEvent(3X11)
Guide to the Xlib Library

XOpenDisplay (3X11)

Name

XOpenDisplay, XCloseDisplay – connect or disconnect to X server

Syntax

Display *XOpenDisplay(*display_name*)
    char **display_name*;

XCloseDisplay(*display*)
    Display **display*;

Arguments

display	Specifies the connection to the X server.
display_name	Specifies the hardware display name, which determines the display and communications domain to be used. On a UNIX-based system, if the display_name is NULL, it defaults to the value of the DISPLAY environment variable.

Description

The `XOpenDisplay` function returns a `Display` structure that serves as the connection to the X server and that contains all the information about that X server. `XOpenDisplay` connects your application to the X server through TCP, UNIX domain, or DECnet communications protocols. If the hostname is a host machine name and a single colon (:) separates the hostname and display number, `XOpenDisplay` connects using TCP streams. If the hostname is *unix* and a single colon (:) separates it from the display number, `XOpenDisplay` connects using UNIX domain IPC streams. If the hostname is not specified, Xlib uses whatever it believes is the fastest transport. If the hostname is a host machine name and a double colon (::) separates the hostname and display number, `XOpenDisplay` connects using DECnet. A single X server can support any or all of these transport mechanisms simultaneously. A particular Xlib implementation can support many more of these transport mechanisms.

If successful, `XOpenDisplay` returns a pointer to a `Display` structure, which is defined in <`X11/Xlib.h`>. If `XOpenDisplay` does not succeed, it returns NULL. After a successful call to `XOpenDisplay`, all of the screens in the display can be used by the client. The screen number specified in the display_name argument is returned by the `DefaultScreen` macro (or the `XDefaultScreen` function). You can access elements of the `Display` and `Screen` structures only by using the information macros or

functions. For information about using macros and functions to obtain information from the Display structure, see section 2.2.1.

The XCloseDisplay function closes the connection to the X server for the display specified in the Display structure and destroys all windows, resource IDs (Window, Font, Pixmap, Colormap, Cursor, and GContext), or other resources that the client has created on this display, unless the close-down mode of the resource has been changed (see XSetCloseDownMode). Therefore, these windows, resource IDs, and other resources should never be referenced again or an error will be generated. Before exiting, you should call XCloseDisplay explicitly so that any pending errors are reported as XCloseDisplay performs a final XSync operation.

XCloseDisplay can generate a BadGC error.

See Also

Guide to the Xlib Library

XParseGeometry (3X11)

Name

XParseGeometry, XGeometry, XParseColor – parse window geometry and color

Syntax

int XParseGeometry(*parsestring*, *x_return*, *y_return*, *width_return*, *height_return*)
    char **parsestring*;
    int **x_return*, **y_return*;
    int **width_return*, **height_return*;

int XGeometry(*display*, *screen*, *position*, *default_position*, *bwidth*, *fwidth*, *fheight*, *xadder*, *yadder*, *x_return*, *y_return*, *width_return*, *height_return*)
    Display **display*;
    int *screen*;
    char **position*, **default_position*;
    unsigned int *bwidth*;
    unsigned int *fwidth*, *fheight*;
    int *xadder*, *yadder*;
    int **x_return*, **y_return*;
    int **width_return*, **height_return*;

Status XParseColor(*display*, *colormap*, *spec*, *exact_def_return*)
    Display **display*;
    Colormap *colormap*;
    char **spec*;
    XColor **exact_def_return*;

Arguments

bwidth Specifies the border width.

colormap Specifies the colormap.

position
default_position
Specify the geometry specifications.

display Specifies the connection to the X server.

exact_def_return
Returns the exact color value for later use and sets the `DoRed`, `DoGreen`, and `DoBlue` flags.

fheight

fwidth	Specify the font height and width in pixels (increment size).
parsestring	Specifies the string you want to parse.
screen	Specifies the screen.
spec	Specifies the color name string; case is ignored.
width_return *height_return*	Return the width and height determined.
xadder *yadder*	Specify additional interior padding needed in the window.
x_return *y_return*	Return the x and y offsets.

Description

By convention, X applications use a standard string to indicate window size and placement. `XParseGeometry` makes it easier to conform to this standard because it allows you to parse the standard window geometry. Specifically, this function lets you parse strings of the form:

[=][<*width*>x<*height*>][{+-}<*xoffset*>{+-}<*yoffset*>]

The items in this form map into the arguments associated with this function. (Items enclosed in <> are integers, items in [] are optional, and items enclosed in { } indicate "choose one of". Note that the brackets should not appear in the actual string.)

The `XParseGeometry` function returns a bitmask that indicates which of the four values (width, height, xoffset, and yoffset) were actually found in the string and whether the x and y values are negative. By convention, –0 is not equal to +0, because the user needs to be able to say "position the window relative to the right or bottom edge." For each value found, the corresponding argument is updated. For each value not found, the argument is left unchanged. The bits are represented by `XValue`, `YValue`, `WidthValue`, `HeightValue`, `XNegative`, or `YNegative` and are defined in <`X11/Xutil.h`>. They will be set whenever one of the values is defined or one of the signs is set.

If the function returns either the `XValue` or `YValue` flag, you should place the window at the requested position.

You pass in the border width (bwidth), size of the increments fwidth and fheight (typically font width and height), and any additional interior space (xadder and yadder) to make it easy to compute the resulting size. The `XGeometry` function returns the position the window should be placed

given a position and a default position. XGeometry determines the placement of a window using a geometry specification as specified by XParseGeometry and the additional information about the window. Given a fully qualified default geometry specification and an incomplete geometry specification, XParseGeometry returns a bitmask value as defined above in the XParseGeometry call, by using the position argument.

The returned width and height will be the width and height specified by default_position as overridden by any user-specified position. They are not affected by fwidth, fheight, xadder, or yadder. The x and y coordinates are computed by using the border width, the screen width and height, padding as specified by xadder and yadder, and the fheight and fwidth times the width and height from the geometry specifications.

The XParseColor function provides a simple way to create a standard user interface to color. It takes a string specification of a color, typically from a command line or XGetDefault option, and returns the corresponding red, green, and blue values that are suitable for a subsequent call to XAllocColor or XStoreColor. The color can be specified either as a color name (as in XAllocNamedColor) or as an initial sharp sign character followed by a numeric specification, in one of the following formats:

#RGB	(4 bits each)
#RRGGBB	(8 bits each)
#RRRGGGBBB	(12 bits each)
#RRRRGGGGBBBB	(16 bits each)

The R, G, and B represent single hexadecimal digits (both uppercase and lowercase). When fewer than 16 bits each are specified, they represent the most-significant bits of the value. For example, #3a7 is the same as #3000a0007000. The colormap is used only to determine which screen to look up the color on. For example, you can use the screen's default colormap.

If the initial character is a sharp sign but the string otherwise fails to fit the above formats or if the initial character is not a sharp sign and the named color does not exist in the server's database, XParseColor fails and returns zero.

XParseColor can generate a BadColor error.

Diagnostics

`BadColor` A value for a Colormap argument does not name a defined Colormap.

See Also

Guide to the Xlib Library

XPolygonRegion (3X11)

Name

XPolygonRegion, XClipBox – generate regions

Syntax

Region XPolygonRegion(*points*, *n*, *fill_rule*)
    XPoint *points[]*;
    int *n*;
    int *fill_rule*;

XClipBox(*r*, *rect_return*)
    Region *r*;
    XRectangle **rect_return*;

Arguments

fill_rule	Specifies the fill-rule you want to set for the specified GC. You can pass `EvenOddRule` or `WindingRule`.
n	Specifies the number of points in the polygon.
points	Specifies an array of points.
r	Specifies the region.
rect_return	Returns the smallest enclosing rectangle.

Description

The `XPolygonRegion` function returns a region for the polygon defined by the points array. For an explanation of fill_rule, see `XCreateGC`.

The `XClipBox` function returns the smallest rectangle enclosing the specified region.

See Also

Guide to the Xlib Library

Name

XPutBackEvent – put events back on the queue

Syntax

XPutBackEvent(*display*, *event*)
Display **display*;
XEvent **event*;

Arguments

display	Specifies the connection to the X server.
event	Specifies a pointer to the event.

Description

The `XPutBackEvent` function pushes an event back onto the head of the display's event queue by copying the event into the queue. This can be useful if you read an event and then decide that you would rather deal with it later. There is no limit to the number of times in succession that you can call `XPutBackEvent`.

See Also

XIfEvent(3X11), XNextEvent(3X11), XSendEvent(3X11)
Guide to the Xlib Library

XPutImage (3X11)

Name

XPutImage, XGetImage, XGetSubImage – transfer images

Syntax

XPutImage(*display*, *d*, *gc*, *image*, *src_x*, *src_y*, *dest_x*, *dest_y*, *width*, *height*)
Display **display*;
Drawable *d*;
GC *gc*;
XImage **image*;
int *src_x*, *src_y*;
int *dest_x*, *dest_y*;
unsigned int *width*, *height*;

XImage *XGetImage(*display*, *d*, *x*, *y*, *width*, *height*, *plane_mask*, *format*)
Display **display*;
Drawable *d*;
int *x*, *y*;
unsigned int *width*, *height*;
long *plane_mask*;
int *format*;

XImage *XGetSubImage(*display*, *d*, *x*, *y*, *width*, *height*, *plane_mask*, *format*, *dest_image*, *dest_x*, *dest_y*)
Display **display*;
Drawable *d*;
int *x*, *y*;
unsigned int *width*, *height*;
unsigned long *plane_mask*;
int *format*;
XImage **dest_image*;
int *dest_x*, *dest_y*;

Arguments

d	Specifies the drawable.
dest_image	Specify the destination image.
dest_x *dest_y*	Specify the x and y coordinates, which are relative to the origin of the drawable and are the coordinates of the subimage or which are relative to the origin of the

	destination rectangle, specify its upper-left corner, and determine where the subimage is placed in the destination image.
display	Specifies the connection to the X server.
format	Specifies the format for the image. You can pass `XYBitmap`, `XYPixmap`, or `ZPixmap`.
gc	Specifies the GC.
image	Specifies the image you want combined with the rectangle.
plane_mask	Specifies the plane mask.
src_x	Specifies the offset in X from the left edge of the image defined by the `XImage` data structure.
src_y	Specifies the offset in Y from the top edge of the image defined by the `XImage` data structure.
width *height*	Specify the width and height of the subimage, which define the dimensions of the rectangle.
x *y*	Specify the x and y coordinates, which are relative to the origin of the drawable and define the upper-left corner of the rectangle.

Description

The `XPutImage` function combines an image in memory with a rectangle of the specified drawable. If `XYBitmap` format is used, the depth must be one, or a `BadMatch` error results. The foreground pixel in the GC defines the source for the one bits in the image, and the background pixel defines the source for the zero bits. For `XYPixmap` and `ZPixmap`, the depth must match the depth of the drawable, or a `BadMatch` error results. The section of the image defined by the src_x, src_y, width, and height arguments is drawn on the specified part of the drawable.

This function uses these GC components: function, plane-mask, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. It also uses these GC mode-dependent components: foreground and background.

`XPutImage` can generate `BadDrawable`, `BadGC`, `BadMatch`, and `BadValue` errors.

The XGetImage function returns a pointer to an XImage structure. This structure provides you with the contents of the specified rectangle of the drawable in the format you specify. If the format argument is XYPixmap, the image contains only the bit planes you passed to the plane_mask argument. If the plane_mask argument only requests a subset of the planes of the display, the depth of the returned image will be the number of planes requested. If the format argument is ZPixmap, XGetImage returns as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits.

XGetImage returns the depth of the image to the depth member of the XImage structure. The depth of the image is as specified when the drawable was created, except when getting a subset of the planes in XYPixmap format, when the depth is given by the number of bits set to 1 in plane_mask.

If the drawable is a pixmap, the given rectangle must be wholly contained within the pixmap, or a BadMatch error results. If the drawable is a window, the window must be viewable, and it must be the case that if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window, or a BadMatch error results. Note that the borders of the window can be included and read with this request. If the window has backing-store, the backing-store contents are returned for regions of the window that are obscured by noninferior windows. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window's depth are also undefined. The pointer cursor image is not included in the returned contents.

XGetImage can generate BadDrawable, BadMatch, and BadValue errors.

The XGetSubImage function updates dest_image with the specified subimage in the same manner as XGetImage. If the format argument is XYPixmap, the image contains only the bit planes you passed to the plane_mask argument. If the format argument is ZPixmap, XGetSubImage returns as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits. As a convenience, XGetSubImage returns a pointer to the same XImage structure specified by dest_image.

The depth of the destination XImage structure must be the same as that of the drawable. If the specified subimage does not fit at the specified location on the destination image, the right and bottom edges are clipped. If the drawable is a pixmap, the given rectangle must be wholly contained within the pixmap, or a BadMatch error results. If the drawable is a window, the window must be viewable, and it must be the case that if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window, or a BadMatch error results. If the window has backing-store, then the backing-store contents are returned for regions of the window that are obscured by noninferior windows. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window's depth are also undefined.

XGetSubImage can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

Diagnostics

BadDrawable
: A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC
: A value for a GContext argument does not name a defined GContext.

BadMatch
: An InputOnly window is used as a Drawable.

BadMatch
: Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue
: Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

XQueryBestSize (3X11)

Name

XQueryBestSize, XQueryBestTile, XQueryBestStipple – determine efficient sizes

Syntax

Status XQueryBestSize(*display*, *class*, *which_screen*, *width*, *height*, *width_return*, *height_return*)
Display **display*;
int *class*;
Drawable *which_screen*;
unsigned int *width*, *height*;
unsigned int **width_return*, **height_return*;

Status XQueryBestTile(*display*, *which_screen*, *width*, *height*, *width_return*, *height_return*)
Display **display*;
Drawable *which_screen*;
unsigned int *width*, *height*;
unsigned int **width_return*, **height_return*;

Status XQueryBestStipple(*display*, *which_screen*, *width*, *height*, *width_return*, *height_return*)
Display **display*;
Drawable *which_screen*;
unsigned int *width*, *height*;
unsigned int **width_return*, **height_return*;

Arguments

class	Specifies the class that you are interested in. You can pass `TileShape`, `CursorShape`, or `StippleShape`.
display	Specifies the connection to the X server.
width *height*	Specify the width and height.
which_screen	Specifies any drawable on the screen.
width_return *height_return*	Return the width and height of the object best supported by the display hardware.

Description

The XQueryBestSize function returns the best or closest size to the specified size. For CursorShape, this is the largest size that can be fully displayed on the screen specified by which_screen. For TileShape, this is the size that can be tiled fastest. For StippleShape, this is the size that can be stippled fastest. For CursorShape, the drawable indicates the desired screen. For TileShape and StippleShape, the drawable indicates the screen and possibly the window class and depth. An InputOnly window cannot be used as the drawable for TileShape or StippleShape, or a BadMatch error results.

XQueryBestSize can generate BadDrawable, BadMatch, and BadValue errors.

The XQueryBestTile function returns the best or closest size, that is, the size that can be tiled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. If an InputOnly window is used as the drawable, a BadMatch error results.

XQueryBestTile can generate BadDrawable and BadMatch errors.

XQueryBestTile can generate BadDrawable and BadMatch errors.

The XQueryBestStipple function returns the best or closest size, that is, the size that can be stippled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. If an InputOnly window is used as the drawable, a BadMatch error results.

XQueryBestStipple can generate BadDrawable and BadMatch errors.

Diagnostics

BadMatch	An InputOnly window is used as a Drawable.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadMatch	The values do not exist for an InputOnly window.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XCreateGC(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

Name

XQueryColor, XQueryColors, XLookupColor – obtain color values

Syntax

XQueryColor(*display*, *colormap*, *def_in_out*)
Display **display*;
Colormap *colormap*;
XColor **def_in_out*;

XQueryColors(*display*, *colormap*, *defs_in_out*, *ncolors*)
Display **display*;
Colormap *colormap*;
XColor *defs_in_out*[];
int *ncolors*;

Status XLookupColor(*display*, *colormap*, *color_name*, *exact_def_return*, *screen_def_return*)
Display **display*;
Colormap *colormap*;
char **color_name*;
XColor **exact_def_return*, **screen_def_return*;

Arguments

colormap Specifies the colormap.

color_name Specifies the color name string (for example, red) whose color definition structure you want returned.

def_in_out Specifies and returns the RGB values for the pixel specified in the structure.

defs_in_out Specifies and returns an array of color definition structures for the pixel specified in the structure.

display Specifies the connection to the X server.

exact_def_return
Returns the exact RGB values.

ncolors Specifies the number of `XColor` structures in the color definition array.

screen_def_return
Returns the closest RGB values provided by the hardware.

XQueryColor (3X11)

Description

The XQueryColor function returns the RGB values for each pixel in the XColor structures and sets the DoRed, DoGreen, and DoBlue flags. The XQueryColors function returns the RGB values for each pixel in the XColor structures and sets the DoRed, DoGreen, and DoBlue flags.

XQueryColor XQueryColors and can generate BadColor and BadValue errors.

The XLookupColor function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified colormap. You should use the ISO Latin-1 encoding; uppercase and lowercase do not matter. XLookupColor returns nonzero if the name existed in the color database or zero if it did not exist.

Diagnostics

BadColor	A value for a Colormap argument does not name a defined Colormap.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XAllocColor(3X11), XCreateColormap(3X11), XStoreColors(3X11)
Guide to the Xlib Library

Name

XQueryPointer – get pointer coordinates

Syntax

Bool XQueryPointer(*display*, *w*, *root_return*, *child_return*, *root_x_return*, *root_y_return*, *win_x_return*, *win_y_return*, *mask_return*)
Display **display*;
Window *w*;
Window **root_return*, **child_return*;
int **root_x_return*, **root_y_return*;
int **win_x_return*, **win_y_return*;
unsigned int **mask_return*;

Arguments

child_return	Returns the child window that the pointer is located in, if any.
display	Specifies the connection to the X server.
mask_return	Returns the current state of the modifier keys and pointer buttons.
root_return	Returns the root window that the pointer is in.
root_x_return *root_y_return*	Return the pointer coordinates relative to the root window's origin.
w	Specifies the window.
win_x_return *win_y_return*	Return the pointer coordinates relative to the specified window.

Description

The `XQueryPointer` function returns the root window the pointer is logically on and the pointer coordinates relative to the root window's origin. If `XQueryPointer` returns `False`, the pointer is not on the same screen as the specified window, and `XQueryPointer` returns `None` to child_return and zero to win_x_return and win_y_return. If `XQueryPointer` returns `True`, the pointer coordinates returned to win_x_return and win_y_return are relative to the origin of the specified window. In this case, `XQueryPointer` returns the child that contains the

pointer, if any, or else None to child_return.

XQueryPointer returns the current logical state of the keyboard buttons and the modifier keys in mask_return. It sets mask_return to the bitwise inclusive OR of one or more of the button or modifier key bitmasks to match the current state of the mouse buttons and the modifier keys.

XQueryPointer can generate a BadWindow error.

Diagnostics

BadWindow	A value for a Window argument does not name a defined Window.

See Also

XGetWindowAttributes(3X11), XQueryTree(3X11)
Guide to the Xlib Library

Name

XQueryTree – query window tree information

Syntax

Status XQueryTree(*display*, *w*, *root_return*, *parent_return*, *children_return*, *nchildren_return*)
Display **display*;
Window *w*;
Window **root_return*;
Window **parent_return*;
Window ***children_return*;
unsigned int **nchildren_return*;

Arguments

children_return
Returns a pointer to the list of children.

display Specifies the connection to the X server.

nchildren_return
Returns the number of children.

parent_return Returns the parent window.

root_return Returns the root window.

w Specifies the window whose list of children, root, parent, and number of children you want to obtain.

Description

The `XQueryTree` function returns the root ID, the parent window ID, a pointer to the list of children windows, and the number of children in the list for the specified window. The children are listed in current stacking order, from bottommost (first) to topmost (last). `XQueryTree` returns zero if it fails and nonzero if it succeeds. To free this list when it is no longer needed, use `XFree`.

Bugs

This really should return a screen *, not a root window ID.

See Also

XGetWindowAttributes(3X11), XQueryPointer(3X11)
Guide to the Xlib Library

Name

XRaiseWindow, XLowerWindow, XCirculateSubwindows, XCirculateSubwindowsUp, XCirculateSubwindowsDown, XRestackWindows – change window stacking order

Syntax

XRaiseWindow(*display*, *w*)
Display **display*;
Window *w*;

XLowerWindow(*display*, *w*)
Display **display*;
Window *w*;

XCirculateSubwindows(*display*, *w*, *direction*)
Display **display*;
Window *w*;
int *direction*;

XCirculateSubwindowsUp(*display*, *w*)
Display **display*;
Window *w*;

XCirculateSubwindowsDown(*display*, *w*)
Display **display*;
Window *w*;

XRestackWindows(*display*, *windows*, *nwindows*);
Display **display*;
Window *windows*[];
int *nwindows*;

Arguments

direction	Specifies the direction (up or down) that you want to circulate the window. You can pass `RaiseLowest` or `LowerHighest`.
display	Specifies the connection to the X server.
nwindows	Specifies the number of windows to be restacked.
w	Specifies the window.
windows	Specifies an array containing the windows to be restacked.

Description

The XRaiseWindow function raises the specified window to the top of the stack so that no sibling window obscures it. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack but leaving its x and y location on the desk constant. Raising a mapped window may generate Expose events for the window and any mapped subwindows that were formerly obscured.

If the override-redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no processing is performed. Otherwise, the window is raised.

XRaiseWindow can generate a BadWindow error.

The XLowerWindow function lowers the specified window to the bottom of the stack so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack but leaving its x and y location on the desk constant. Lowering a mapped window will generate Expose events on any windows it formerly obscured.

If the override-redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no processing is performed. Otherwise, the window is lowered to the bottom of the stack.

XLowerWindow can generate a BadWindow error.

The XCirculateSubwindows function circulates children of the specified window in the specified direction. If you specify RaiseLowest, XCirculateSubwindows raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. If you specify LowerHighest, XCirculateSubwindows lowers the highest mapped child (if any) that occludes another child to the bottom of the stack. Exposure processing is then performed on formerly obscured windows. If some other client has selected SubstructureRedirectMask on the window, the X server generates a CirculateRequest event, and no further processing is performed. If a child is actually restacked, the X server generates a CirculateNotify event.

XCirculateSubwindows can generate BadValue and BadWindow errors.

The XCirculateSubwindowsUp function raises the lowest mapped child of the specified window that is partially or completely occluded by another child. Completely unobscured children are not affected. This is a convenience function equivalent to XCirculateSubwindows with RaiseLowest specified.

XCirculateSubwindowsUp can generate a BadWindow error.

The XCirculateSubwindowsDown function lowers the highest mapped child of the specified window that partially or completely occludes another child. Completely unobscured children are not affected. This is a convenience function equivalent to XCirculateSubwindows with LowerHighest specified.

XCirculateSubwindowsDown can generate a BadWindow error.

The XRestackWindows function restacks the windows in the order specified, from top to bottom. The stacking order of the first window in the windows array is unaffected, but the other windows in the array are stacked underneath the first window, in the order of the array. The stacking order of the other windows is not affected. For each window in the window array that is not a child of the specified window, a BadMatch error results.

If the override-redirect attribute of a window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates ConfigureRequest events for each window whose override-redirect flag is not set, and no further processing is performed. Otherwise, the windows will be restacked in top to bottom order.

XRestackWindows can generate BadWindow error.

Diagnostics

BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XChangeWindowAttributes(3X11), XConfigureWindow(3X11), XCreateWindow(3X11), XDestroyWindow(3X11), XMapWindow(3X11), XUnmapWindow(3X11)
Guide to the Xlib Library

Name

XReadBitmapFile, XWriteBitmapFile, XCreatePixmapFromBitmapData, XCreateBitmapFromData – manipulate bitmaps

Syntax

int XReadBitmapFile(*display*, *d*, *filename*, *width_return*, *height_return*, *bitmap_return*, *x_hot_return*, *y_hot_return*)
Display **display*;
Drawable *d*;
char **filename*;
unsigned int **width_return*, **height_return*;
Pixmap **bitmap_return*;
int **x_hot_return*, **y_hot_return*;

int XWriteBitmapFile(*display*, *filename*, *bitmap*, *width*, *height*, *x_hot*, *y_hot*)
Display **display*;
char **filename*;
Pixmap *bitmap*;
unsigned int *width*, *height*;
int *x_hot*, *y_hot*;

Pixmap XCreatePixmapFromBitmapData(*display*, *d*, *data*, *width*, *height*, *fg*, *bg*, *depth*)
Display **display*;
Drawable *d*;
char **data*;
unsigned int *width*, *height*;
unsigned long *fg*, *bg*;
unsigned int *depth*;

Pixmap XCreateBitmapFromData(*display*, *d*, *data*, *width*, *height*)
Display **display*;
Drawable *d*;
char **data*;
unsigned int *width*, *height*;

Arguments

bitmap	Specifies the bitmap.
bitmap_return	Returns the bitmap that is created.
d	Specifies the drawable that indicates the screen.

data	Specifies the data in bitmap format.
data	Specifies the location of the bitmap data.
depth	Specifies the depth of the pixmap.
display	Specifies the connection to the X server.
fg *bg*	Specify the foreground and background pixel values to use.
filename	Specifies the file name to use. The format of the file name is operating-system dependent.
width *height*	Specify the width and height.
width_return *height_return*	Return the width and height values of the read in bitmap file.
x_hot *y_hot*	Specify where to place the hotspot coordinates (or -1,-1 if none are present) in the file.
x_hot_return *y_hot_return*	Return the hotspot coordinates.

Description

The `XReadBitmapFile` function reads in a file containing a bitmap. The file can be either in the standard X version 10 format (that is, the format used by X version 10 bitmap program) or in the X version 11 bitmap format. If the file cannot be opened, `XReadBitmapFile` returns `BitmapOpenFailed`. If the file can be opened but does not contain valid bitmap data, it returns `BitmapFileInvalid`. If insufficient working storage is allocated, it returns `BitmapNoMemory`. If the file is readable and valid, it returns `BitmapSuccess`.

`XReadBitmapFile` returns the bitmap's height and width, as read from the file, to width_return and height_return. It then creates a pixmap of the appropriate size, reads the bitmap data from the file into the pixmap, and assigns the pixmap to the caller's variable bitmap. The caller must free the bitmap using `XFreePixmap` when finished. If *name*_x_hot and *name*_y_hot exist, `XReadBitmapFile` returns them to x_hot_return and y_hot_return; otherwise, it returns –1,–1.

`XReadBitmapFile` can generate `BadAlloc` and `BadDrawable` errors.

The XWriteBitmapFile function writes a bitmap out to a file. While XReadBitmapFile can read in either X version 10 format or X version 11 format, XWriteBitmapFile always writes out X version 11 format. If the file cannot be opened for writing, it returns BitmapOpenFailed. If insufficient memory is allocated, XWriteBitmapFile returns BitmapNoMemory; otherwise, on no error, it returns BitmapSuccess. If x_hot and y_hot are not –1, –1, XWriteBitmapFile writes them out as the hotspot coordinates for the bitmap.

XWriteBitmapFile can generate BadDrawable and BadMatch errors.

The XCreatePixmapFromBitmapData function creates a pixmap of the given depth and then does a bitmap-format XPutImage of the data into it. The depth must be supported by the screen of the specified drawable, or a BadMatch error results.

XCreatePixmapFromBitmapData can generate BadAlloc and BadMatch errors.

The XCreateBitmapFromData function allows you to include in your C program (using #include) a bitmap file that was written out by XWriteBitmapFile (X version 11 format only) without reading in the bitmap file. The following example creates a gray bitmap:

#include "gray.bitmap"

Pixmap bitmap;
bitmap = XCreateBitmapFromData(display, window, gray_bits, gray_width, gray_height

If insufficient working storage was allocated, XCreateBitmapFromData returns None. It is your responsibility to free the bitmap using XFreePixmap when finished.

XCreateBitmapFromData can generate a BadAlloc error.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadMatch	An InputOnly window is used as a Drawable.

See Also

Guide to the Xlib Library

Name

XRecolorCursor, XFreeCursor, XQueryBestCursor – manipulate cursors

Syntax

XRecolorCursor(*display*, *cursor*, *foreground_color*, *background_color*)
Display **display*;
Cursor *cursor*;
XColor **foreground_color*, **background_color*;

XFreeCursor(*display*, *cursor*)
Display **display*;
Cursor *cursor*;

Status XQueryBestCursor(*display*, *d*, *width*, *height*, *width_return*, *height_return*)
Display **display*;
Drawable *d*;
unsigned int *width*, *height*;
unsigned int **width_return*, **height_return*;

Arguments

background_color
Specifies the RGB values for the background of the source.

cursor Specifies the cursor.

d Specifies the drawable, which indicates the screen.

display Specifies the connection to the X server.

foreground_color
Specifies the RGB values for the foreground of the source.

width
height Specify the width and heightof the cursor that you want the size information for.

width_return
height_return Return the best width and height that is closest to the specified width and height.

XRecolorCursor (3X11)

Description

The `XRecolorCursor` function changes the color of the specified cursor, and if the cursor is being displayed on a screen, the change is visible immediately.

`XRecolorCursor` can generate a `BadCursor` error.

The `XFreeCursor` function deletes the association between the cursor resource ID and the specified cursor. The cursor storage is freed when no other resource references it. The specified cursor ID should not be referred to again.

`XFreeCursor` can generate a `BadCursor` error.

Some displays allow larger cursors than other displays. The `XQueryBestCursor` function provides a way to find out what size cursors are actually possible on the display. It returns the largest size that can be displayed. Applications should be prepared to use smaller cursors on displays that cannot support large ones.

`XQueryBestCursor` can generate a `BadDrawable` error.

Diagnostics

`BadCursor` A value for a Cursor argument does not name a defined Cursor.

`BadDrawable`
A value for a Drawable argument does not name a defined Window or Pixmap.

See Also

XCreateFontCursor(3X11), XDefineCusor(3X11)
Guide to the Xlib Library

Name

XReparentWindow – reparent windows

Syntax

XReparentWindow(*display*, *w*, *parent*, *x*, *y*)
Display **display*;
Window *w*;
Window *parent*;
int *x*, *y*;

Arguments

display	Specifies the connection to the X server.
parent	Specifies the parent window.
w	Specifies the window.
x *y*	Specify the x and y coordinatesof the position in the new parent window.

Description

If the specified window is mapped, `XReparentWindow` automatically performs an `UnmapWindow` request on it, removes it from its current position in the hierarchy, and inserts it as the child of the specified parent. The window is placed in the stacking order on top with respect to sibling windows.

After reparenting the specified window, `XReparentWindow` causes the X server to generate a `ReparentNotify` event. The override_redirect member returned in this event is set to the window's corresponding attribute. Window manager clients usually should ignore this window if this member is set to `True`. Finally, if the specified window was originally mapped, the X server automatically performs a `MapWindow` request on it.

The X server performs normal exposure processing on formerly obscured windows. The X server might not generate `Expose` events for regions from the initial `UnmapWindow` request that are immediately obscured by the final `MapWindow` request. A `BadMatch` error results if:

- The new parent window is not on the same screen as the old parent window.

- The new parent window is the specified window or an inferior of the specified window.
- The specified window has a `ParentRelative` background, and the new parent window is not the same depth as the specified window.

`XReparentWindow` can generate `BadMatch` and `BadWindow` errors.

Diagnostics

`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XChangeSaveSet(3X11)
Guide to the Xlib Library

Name

XrmGetResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource – retrieve database resources and search lists

Syntax

Bool XrmGetResource(*database*, *str_name*, *str_class*, *str_type_return*, *value_return*)
XrmDatabase *database*;
char **str_name*;
char **str_class*;
char ***str_type_return*;
XrmValue **value_return*;

Bool XrmQGetResource(*database*, *quark_name*, *quark_class*, *quark_type_return*, *value_return*)
XrmDatabase *database*;
XrmNameList *quark_name*;
XrmClassList *quark_class*;
XrmRepresentation **quark_type_return*;
XrmValue **value_return*;

typedef XrmHashTable *XrmSearchList;

Bool XrmQGetSearchList(*database*, *names*, *classes*, *list_return*, *list_length*)
XrmDatabase *database*;
XrmNameList *names*;
XrmClassList *classes*;
XrmSearchList *list_return*;
int *list_length*;

Bool XrmQGetSearchResource(*list*, *name*, *class*, *type_return*, *value_return*)
XrmSearchList *list*;
XrmName *name*;
XrmClass *class*;
XrmRepresentation **type_return*;
XrmValue **value_return*;

Arguments

class	Specifies the resource class.
classes	Specifies a list of resource classes.

database Specifies the database that is to be used.

list Specifies the search list returned by `XrmQGetSearchList`.

list_length Specifies the number of entries (not the byte size) allocated for list_return.

list_return Returns a search list for further use.

name Specifies the resource name.

names Specifies a list of resource names.

quark_class Specifies the fully qualified class of the value being retrieved (as a quark).

quark_name Specifies the fully qualified name of the value being retrieved (as a quark).

quark_type_return Returns a pointer to the representation type of the destination (as a quark).

str_class Specifies the fully qualified class of the value being retrieved (as a string).

str_name Specifies the fully qualified name of the value being retrieved (as a string).

str_type_return Returns a pointer to the representation type of the destination (as a string).

type_return Returns data representation type.

value_return Returns the value in the database.

Description

The `XrmGetResource` and `XrmQGetResource` functions retrieve a resource from the specified database. Both take a fully qualified name/class pair, a destination resource representation, and the address of a value (size/address pair). The value and returned type point into database memory; therefore, you must not modify the data.

The database only frees or overwrites entries on `XrmPutResource`, `XrmQPutResource`, or `XrmMergeDatabases`. A client that is not storing new values into the database or is not merging the database should be safe using the address passed back at any time until it exits. If a resource was found, both `XrmGetResource` and `XrmQGetResource` return `True`; otherwise, they return `False`.

The XrmQGetSearchList function takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order and uses the same algorithm as XrmGetResource for determining precedence. If list_return was large enough for the search list, XrmQGetSearchList returns True; otherwise, it returns False.

The size of the search list that the caller must allocate is dependent upon the number of levels and wildcards in the resource specifiers that are stored in the database. The worst case length is 3^n, where n is the number of name or class components in names or classes.

When using XrmQGetSearchList followed by multiple probes for resources with a common name and class prefix, only the common prefix should be specified in the name and class list to XrmQGetSearchList.

The XrmQGetSearchResource function searches the specified database levels for the resource that is fully identified by the specified name and class. The search stops with the first match. XrmQGetSearchResource returns True if the resource was found; otherwise, it returns False.

A call to XrmQGetSearchList with a name and class list containing all but the last component of a resource name followed by a call to XrmQGetSearchResource with the last component name and class returns the same database entry as XrmGetResource and XrmQGetResource with the fully qualified name and class.

See Also

XrmInitialize(3X11), XrmMergeDatabases(3X11), XrmPutResource(3X11), XrmUniqueQuark(3X11)
Guide to the Xlib Library

XrmInitialize (3X11)

Name

XrmInitialize, XrmParseCommand – initialize the Resource Manager and parse the command line

Syntax

void XrmInitialize( );

void XrmParseCommand(*database*, *table*, *table_count*, *name*, *argc_in_out*, *argv_in_out*,)
XrmDatabase **database*;
XrmOptionDescList *table*;
int *table_count*;
char **name*;
int **argc_in_out*;
char ***argv_in_out*;

Arguments

argc_in_out	Specifies the number of arguments and returns the number of remaining arguments.
argv_in_out	Specifies a pointer to the command line arguments and returns the remaining arguments.
database	Specifies a pointer to the resource database.
name	Specifies the application name.
table	Specifies the table of command line arguments to be parsed.
table_count	Specifies the number of entries in the table.

Description

The `XrmInitialize` function initialize the resource manager.

The `XrmParseCommand` function parses an (argc, argv) pair according to the specified option table, loads recognized options into the specified database with type "String," and modifies the (argc, argv) pair to remove all recognized options.

The specified table is used to parse the command line. Recognized entries in the table are removed from argv, and entries are made in the specified resource database. The table entries contain information on the option string, the option name, the style of option, and a value to provide if the option kind is `XrmoptionNoArg`. The argc argument specifies the number of

arguments in argv and is set to the remaining number of arguments that were not parsed. The name argument should be the name of your application for use in building the database entry. The name argument is prefixed to the resourceName in the option table before storing the specification. No separating (binding) character is inserted. The table must contain either a period (.) or an asterisk (*) as the first character in each resourceName entry. To specify a more completely qualified resource name, the resourceName entry can contain multiple components.

See Also

XrmGetResource(3X11), XrmMergeDatabases(3X11), XrmPutResource(3X11), XrmUniqueQuark(3X11)
Guide to the Xlib Library

XrmMergeDatabases (3X11)

Name

XrmMergeDatabases, XrmGetFileDatabase, XrmPutFileDatabase, XrmGetStringDatabase – manipulate resource databases

Syntax

void XrmMergeDatabases(*source_db*, *target_db*)
    XrmDatabase *source_db*, **target_db*;

XrmDatabase XrmGetFileDatabase(*filename*)
    char **filename*;

void XrmPutFileDatabase(*database*, *stored_db*)
    XrmDatabase *database*;
    char **stored_db*;

XrmDatabase XrmGetStringDatabase(*data*)
    char **data*;

Arguments

data	Specifies the database contents using a string.
database	Specifies the database that is to be used.
filename	Specifies the resource database file name.
source_db	Specifies the resource database that is to be merged into the target database.
stored_db	Specifies the file name for the stored database.
target_db	Specifies a pointer to the resource database into which the source database is to be merged.

Description

The `XrmMergeDatabases` function merges the contents of one database into another. It may overwrite entries in the destination database. This function is used to combine databases (for example, an application specific database of defaults and a database of user preferences). The merge is destructive; that is, the source database is destroyed.

The `XrmGetFileDatabase` function opens the specified file, creates a new resource database, and loads it with the specifications read in from the specified file. The specified file must contain lines in the format accepted by `XrmPutLineResource`. If it cannot open the specified file,

XrmGetFileDatabase returns NULL.

The XrmPutFileDatabase function stores a copy of the specified database in the specified file. The file is an ASCII text file that contains lines in the format that is accepted by XrmPutLineResource.

The XrmGetStringDatabase function creates a new database and stores the resources specified in the specified null-terminated string. XrmGetStringDatabase is similar to XrmGetFileDatabase except that it reads the information out of a string instead of out of a file. Each line is separated by a new-line character in the format accepted by XrmPutLineResource.

See Also

XrmGetResource(3X11), XrmInitialize(3X11), XrmPutResource(3X11), XrmUniqueQuark(3X11)
Guide to the Xlib Library

XrmPutResource (3X11)

Name

XrmPutResource, XrmQPutResource, XrmPutStringResource, XrmQPutStringResource, XrmPutLineResource – store database resources

Syntax

void XrmPutResource(*database*, *specifier*, *type*, *value*)
XrmDatabase **database*;
char **specifier*;
char **type*;
XrmValue **value*;

void XrmQPutResource(*database*, *bindings*, *quarks*, *type*, *value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
XrmRepresentation *type*;
XrmValue **value*;

void XrmPutStringResource(*database*, *specifier*, *value*)
XrmDatabase **database*;
char **specifier*;
char **value*;

void XrmQPutStringResource(*database*, *bindings*, *quarks*, *value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
char **value*;

void XrmPutLineResource(*database*, *line*)
XrmDatabase **database*;
char **line*;

Arguments

bindings	Specifies a list of bindings.
database	Specifies a pointer to the resource database.
line	Specifies the resource value pair as a single string. A single colon (:) separates the name from the value.
quarks	Specifies the complete or partial name or the class list of the resource.

specifier	Specifies a complete or partial specification of the resource.
type	Specifies the type of the resource.
value	Specifies the value of the resource, which is specified as a string.

Description

If database contains NULL, `XrmPutResource` creates a new database and returns a pointer to it. `XrmPutResource` is a convenience function that calls `XrmStringToBindingQuarkList` followed by:

XrmQPutResource(database, bindings, quarks, XrmStringToQuark(type), value)

If database contains NULL, `XrmQPutResource` creates a new database and returns a pointer to it.

If database contains NULL, `XrmPutStringResource` creates a new database and returns a pointer to it. `XrmPutStringResource` adds a resource with the specified value to the specified database. `XrmPutStringResource` is a convenience routine that takes both the resource and value as null-terminated strings, converts them to quarks, and then calls `XrmQPutResource`, using a "String" representation type.

If database contains NULL, `XrmQPutStringResource` creates a new database and returns a pointer to it. `XrmQPutStringResource` is a convenience routine that constructs an `XrmValue` for the value string (by calling `strlen` to compute the size) and then calls `XrmQPutResource`, using a "String" representation type.

If database contains NULL, `XrmPutLineResource` creates a new database and returns a pointer to it. `XrmPutLineResource` adds a single resource entry to the specified database. Any white space before or after the name or colon in the line argument is ignored. The value is terminated by a new-line or a NULL character. To allow values to contain embedded new-line characters, a "\n" is recognized and replaced by a new-line character. For example, line might have the value "xterm*background:green\n". Null-terminated strings without a new line are also permitted.

See Also

XrmGetResource(3X11), XrmInitialize(3X11), XrmMergeDatabases(3X11), XrmUniqueQuark(3X11)
Guide to the Xlib Library

XrmUniqueQuark (3X11)

Name

XrmUniqueQuark, XrmStringToQuark, XrmQuarkToString, XrmStringToQuarkList, XrmStringToBindingQuarkList – manipulate resource quarks

Syntax

XrmQuark XrmUniqueQuark( )

#define XrmStringToName(string) XrmStringToQuark(string) #define XrmStringToClass(string) XrmStringToQuark(string) #define XrmStringToRepresentation(string) XrmStringToQuark(string)

XrmQuark XrmStringToQuark(*string*)
char **string*;

#define XrmNameToString(name) XrmQuarkToString(name) #define XrmClassToString(class) XrmQuarkToString(class) #define XrmRepresentationToString(type) XrmQuarkToString(type)

char *XrmQuarkToString(*quark*)
XrmQuark *quark*;

#define XrmStringToNameList(str, name) XrmStringToQuarkList((str), (name)) #define XrmStringToClassList(str,class) XrmStringToQuarkList((str), (class))

void XrmStringToQuarkList(*string*, *quarks_return*)
char **string*;
XrmQuarkList *quarks_return*;

XrmStringToBindingQuarkList(*string*, *bindings_return*, *quarks_return*)
char **string*;
XrmBindingList *bindings_return*;
XrmQuarkList *quarks_return*;

Arguments

bindings_return	Returns the binding list.
quark	Specifies the quark for which the equivalent string is desired.
quarks_return	Returns the list of quarks.
string	Specifies the string for which a quark is to be allocated.

Description

The `XrmUniqueQuark` function allocates a quark that is guaranteed not to represent any string that is known to the resource manager.

These functions can be used to convert to and from quark representations. The string pointed to by the return value must not be modified or freed. If no string exists for that quark, `XrmQuarkToString` returns NULL.

The `XrmQuarkToString` function converts the specified resource quark representation back to a string.

The `XrmStringToQuarkList` function converts the null-terminated string (generally a fully qualified name) to a list of quarks. The components of the string are separated by a period or asterisk character.

A binding list is a list of type `XrmBindingList` and indicates if components of name or class lists are bound tightly or loosely (that is, if wildcarding of intermediate components is specified).

typedef enum {XrmBindTightly, XrmBindLoosely} XrmBinding, *XrmBindingList

`XrmBindTightly` indicates that a period separates the components, and `XrmBindLoosely` indicates that an asterisk separates the components.

The `XrmStringToBindingQuarkList` function converts the specified string to a binding list and a quark list. Component names in the list are separated by a period or an asterisk character. If the string does not start with period or asterisk, a period is assumed. For example, ''*a.b*c'' becomes:

quarks	a	b	c
bindings	loose	tight	loose

See Also

XrmGetResource(3X11), XrmInitialize(3X11), XrmMergeDatabases(3X11), XrmPutResource(3X11)
Guide to the Xlib Library

XSaveContext (3X11)

Name

XSaveContext, XFindContext, XDeleteContext, XUniqueContext – associative lookup routines

Syntax

int XSaveContext(*display*, *w*, *context*, *data*)
Display **display*;
Window *w*;
XContext *context*;
caddr_t *data*;

int XFindContext(*display*, *w*, *context*, *data_return*)
Display **display*;
Window *w*;
XContext *context*;
caddr_t **data_return*;

int XDeleteContext(*display*, *w*, *context*)
Display **display*;
Window *w*;
XContext *context*;

XContext XUniqueContext()

Arguments

context	Specifies the context type to which the data belongs.
data	Specifies the data to be associated with the window and type.
data_return	Returns a pointer to the data.
display	Specifies the connection to the X server.
w	Specifies the window with which the data is associated.

Description

If an entry with the specified window and type already exists, `XSaveContext` overrides it with the specified context. The `XSaveContext` function returns a nonzero error code if an error has occurred and zero otherwise. Possible errors are `XCNOMEM` (out of memory).

Because it is a return value, the data is a pointer. The `XFindContext` function returns a nonzero error code if an error has occurred and zero otherwise. Possible errors are `XCNOENT` (context-not-found).

The XDeleteContext function deletes the entry for the given window and type from the data structure. This function returns the same error codes that XFindContext returns if called with the same arguments. XDeleteContext does not free the data whose address was saved.

The XUniqueContext function creates a unique context type that may be used in subsequent calls to XSaveContext.

See Also

Guide to the Xlib Library

XSelectInput (3X11)

Name

XSelectInput, XSelectAsyncInput – select input events

Syntax

XSelectInput(*display*, *w*, *event_mask*)
Display **display*;
Window *w*;
long *event_mask*;

XSelectAsyncInput(*display*, *w*, *event_mask*, *procedure*, *argument*)
Display **display*;
Window *w*;
unsigned long *event_mask*;
int (**procedure*)()
unsigned long *argument*;

Arguments

argument	Specifies the argument that is to be passed to the specified procedure.
display	Specifies the connection to the X server.
event_mask	Specifies the event mask.
procedure	Specifies the procedure that is to be called.
w	Specifies the window whose events you are interested in.

Description

The `XSelectInput` function requests that the X server report the events associated with the specified event mask. Initially, X will not report any of these events. Events are reported relative to a window. If a window is not interested in a device event, it usually propagates to the closest ancestor that is interested, unless the do_not_propagate mask prohibits it.

Setting the event-mask attribute of a window overrides any previous call for the same window but not for other clients. Multiple clients can select for the same events on the same window with the following restrictions:

- Multiple clients can select events on the same window because their event masks are disjoint. When the X server generates an event, it reports it to all interested clients.

- Only one client at a time can select `CirculateRequest`, `ConfigureRequest`, or `MapRequest` events, which are associated with the event mask `SubstructureRedirectMask`.
- Only one client at a time can select a `ResizeRequest` event, which is associated with the event mask `ResizeRedirectMask`.
- Only one client at a time can select a `ButtonPress` event, which is associated with the event mask `ButtonPressMask`.

The server reports the event to all interested clients.

`XSelectInput` can generate a `BadWindow` error.

The `XSelectAsyncInput` function establishes asynchrous notification mode and calls the specified procedure, passing it the specified argument when you receive the event selected with the event_mask. Anyone who asynchrous notification cannot also use the SIGIO signal.

Diagnostics

`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

Guide to the Xlib Library

XSendEvent (3X11)

Name

XSendEvent, XDisplayMotionBufferSize, XGetMotionEvents – send events

Syntax

Status XSendEvent(*display*, *w*, *propagate*, *event_mask*, *event_send*)
Display **display*;
Window *w*;
Bool *propagate*;
long *event_mask*;
XEvent **event_send*;

unsigned long XDisplayMotionBufferSize(*display*)
Display **display*;

XTimeCoord *XGetMotionEvents(*display*, *w*, *start*, *stop*, *nevents_return*)
Display **display*;
Window *w*;
Time *start*, *stop*;
int **nevents_return*;

Arguments

display	Specifies the connection to the X server.
event_mask	Specifies the event mask.
event_send	Specifies a pointer to the event that is to be sent.
nevents_return	Returns the number of events from the motion history buffer.
propagate	Specifies a Boolean value.
start *stop*	Specify the time interval in which the events are returned from the motion history buffer. You can pass a timestamp or `CurrentTime. PointerWindow`,
w	Specifies the window the window the event is to be sent to,.

Description

The `XSendEvent` function identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs. This function requires you to pass an event mask. For a discussion of the valid event mask names, see section 8.3. This function uses the w argument to identify the destination window as follows:

- If w is PointerWindow, the destination window is the window that contains the pointer.
- If w is InputFocus and if the focus window contains the pointer, the destination window is the window that contains the pointer; otherwise, the destination window is the focus window.

To determine which clients should receive the specified events, XSendEvent uses the propagate argument as follows:

- If event_mask is the empty set, the event is sent to the client that created the destination window. If that client no longer exists, no event is sent.
- If propagate is False, the event is sent to every client selecting on destination any of the event types in the event_mask argument.
- If propagate is True and no clients have selected on destination any of the event types in event-mask, the destination is replaced with the closest ancestor of destination for which some client has selected a type in event-mask and for which no intervening window has that type in its do-not-propagate-mask. If no such window exists or if the window is an ancestor of the focus window and InputFocus was originally specified as the destination, the event is not sent to any clients. Otherwise, the event is reported to every client selecting on the final destination any of the types specified in event_mask.

The event in the XEvent structure must be one of the core events or one of the events defined by an extension (or a BadValue error results) so that the X server can correctly byte-swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the X server except to force send_event to True in the forwarded event and to set the serial number in the event correctly.

XSendEvent returns zero if the conversion to wire protocol format failed and returns nonzero otherwise. XSendEvent can generate BadValue and BadWindow errors.

The server may retain the recent history of the pointer motion and do so to a finer granularity than is reported by MotionNotify events. The XGetMotionEvents function makes this history available.

The XGetMotionEvents function returns all events in the motion history buffer that fall between the specified start and stop times, inclusive, and that have coordinates that lie within the specified window (including its borders) at its present placement. If the start time is later than the stop time or if the start time is in the future, no events are returned. If the stop time is in the

future, it is equivalent to specifying `CurrentTime`.
`XGetMotionEvents` can generate a `BadWindowe` error.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XIfEvent(3X11), XNextEvent(3X11), XPutBackEvent(3X11)
Guide to the Xlib Library

Name

XSetArcMode, XSetSubwindowMode, XSetGraphicsExposure – GC convenience routines

Syntax

XSetArcMode(*display*, *gc*, *arc_mode*)
Display **display*;
GC *gc*;
int *arc_mode*;

XSetSubwindowMode(*display*, *gc*, *subwindow_mode*)
Display **display*;
GC *gc*;
int *subwindow_mode*;

XSetGraphicsExposures(*display*, *gc*, *graphics_exposures*)
Display **display*;
GC *gc*;
Bool *graphics_exposures*;

Arguments

arc_mode Specifies the arc mode. You can pass `ArcChord` or `ArcPieSlice`.

display Specifies the connection to the X server.

gc Specifies the GC.

graphics_exposures
Specifies a Boolean value that indicates whether you want `GraphicsExpose` and `NoExpose` events to be reported when calling `XCopyArea` and `XCopyPlane` with this GC.

subwindow_mode
Specifies the subwindow mode. You can pass `ClipByChildren` or `IncludeInferiors`.

Description

The `XSetArcMode` function sets the arc mode in the specified GC.

`XSetArcMode` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

XSetArcMode (3X11)

The `XSetSubwindowMode` function sets the subwindow mode in the specified GC.

`XSetSubwindowMode` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

The `XSetGraphicsExposures` function sets the graphics-exposures flag in the specified GC.

`XSetGraphicsExposures` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadGC`	A value for a GContext argument does not name a defined GContext.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

Name

XSetClassHint, XGetClassHint – set or get class hint

Syntax

XSetClassHint(*display*, *w*, *class_hints*)
Display **display*;
Window *w*;
XClassHint **class_hints*;

Status XGetClassHint(*display*, *w*, *class_hints_return*)
Display **display*;
Window *w*;
XClassHint **class_hints_return*;

Arguments

class_hints	Specifies a pointer to a `XClassHint` structure that is to be used.
class_hints_return	Returns the `XClassHint` structure.
display	Specifies the connection to the X server.
w	Specifies the window.

Description

The `XSetClassHint` function sets the class hint for the specified window.

`XSetClassHint` can generate `BadAlloc` and `BadWindow` errors.

The `XGetClassHint` function returns the class of the specified window. To free res_name and res_class when finished with the strings, use `XFree`.

`XGetClassHint` can generate a `BadWindow` error.

Property

WM_CLASS

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined

Window.

See Also

XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetClipOrigin, XSetClipMask, XSetClipRectangles – GC convenience routines

Syntax

XSetClipOrigin(*display*, *gc*, *clip_x_origin*, *clip_y_origin*)
Display **display*;
GC *gc*;
int *clip_x_origin*, *clip_y_origin*;

XSetClipMask(*display*, *gc*, *pixmap*)
Display **display*;
GC *gc*;
Pixmap *pixmap*;

XSetClipRectangles(*display*, *gc*, *clip_x_origin*, *clip_y_origin*, *rectangles*, *n*, *ordering*)
Display **display*;
GC *gc*;
int *clip_x_origin*, *clip_y_origin*;
XRectangle *rectangles*[];
int *n*;
int *ordering*;

Arguments

display	Specifies the connection to the X server.
clip_x_origin *clip_y_origin*	Specify the x and y coordinates of the clip-mask origin.
gc	Specifies the GC.
n	Specifies the number of rectangles.
ordering	Specifies the ordering relations on the rectangles. You can pass `Unsorted`, `YSorted`, `YXSorted`, or `YXBanded`.
pixmap	Specifies the pixmap or `None`.
rectangles	Specifies an array of rectangles that define the clip-mask.

XSetClipOrigin (3X11)

Description

The XSetClipOrigin function sets the clip origin in the specified GC. The clip-mask origin is interpreted relative to the origin of whatever destination drawable is specified in the graphics request.

XSetClipOrigin can generate BadAlloc and BadGC errors.

The XSetClipMask function sets the clip-mask in the specified GC to the specified pixmap. If the clip-mask is set to None, the pixels are are always drawn (regardless of the clip-origin).

XSetClipMask can generate BadAlloc, BadGC, BadMatch, and BadValue errors.

The XSetClipRectangles function changes the clip-mask in the specified GC to the specified list of rectangles and sets the clip origin. The output is clipped to remain contained within the rectangles. The clip-origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The rectangle coordinates are interpreted relative to the clip-origin. The rectangles should be nonintersecting, or the graphics results will be undefined. Note that the list of rectangles can be empty, which effectively disables output. This is the opposite of passing None as the clip-mask in XCreateGC, XChangeGC, and XSetClipMask.

If known by the client, ordering relations on the rectangles can be specified with the ordering argument. This may provide faster operation by the server. If an incorrect ordering is specified, the X server may generate a BadMatch error, but it is not required to do so. If no error is generated, the graphics results are undefined. Unsorted means the rectangles are in arbitrary order. YSorted means that the rectangles are nondecreasing in their Y origin. YXSorted additionally constrains YSorted order in that all rectangles with an equal Y origin are nondecreasing in their X origin. YXBanded additionally constrains YXSorted by requiring that, for every possible Y scanline, all rectangles that include that scanline have an identical Y origins and Y extents.

XSetClipRectangles can generate BadAlloc, BadGC, BadMatch, and BadValue errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadGC	A value for a GContext argument does not name a defined

GContext.

`BadMatch` Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

`BadValue` Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetArcMode(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

XSetCloseDownMode (3X11)

Name

XSetCloseDownMode, XKillClient – control clients

Syntax

XSetCloseDownMode(*display, close_mode*)
    Display **display*;
    int *close_mode*;

XKillClient(*display, resource*)
    Display **display*;
    XID *resource*;

Arguments

close_mode	Specifies the client close-down mode. You can pass `DestroyAll`, `RetainPermanent`, or `RetainTemporary`.
display	Specifies the connection to the X server.
resource	Specifies any resource associated with the client that you want to destroy or `AllTemporary`.

Description

The `XSetCloseDownMode` defines what will happen to the client's resources at connection close. A connection starts in `DestroyAll` mode. For information on what happens to the client's resources when the close_mode argument is `RetainPermanent` or `RetainTemporary`, see section 2.6.

`XSetCloseDownMode` can generate a `BadValue` error.

The `XKillClient` function forces a close-down of the client that created the resource if a valid resource is specified. If the client has already terminated in either `RetainPermanent` or `RetainTemporary` mode, all of the client's resources are destroyed. If `AllTemporary` is specified, the resources of all clients that have terminated in `RetainTemporary` are destroyed (see section 2.6). This permits implementation of window manager facilities that aid debugging. A client can set its close-down mode to `RetainTemporary`. If the client then crashes, its windows would not be destroyed. The programmer can then inspect the application's window tree and use the window manager to destroy the zombie windows.

`XKillClient` can generate a `BadValue` error.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

XSetCommand (3X11)

Name

XSetCommand – set command atom

Syntax

XSetCommand(*display*, *w*, *argv*, *argc*)
Display **display*;
Window *w*;
char ***argv*;
int *argc*;

Arguments

argc	Specifies the number of arguments.
argv	Specifies the application's argument list.
display	Specifies the connection to the X server.
w	Specifies the window.

Description

The `XSetCommand` function sets the command and arguments used to invoke the application. (Typically, argv is the argv array of your main program.)

`XSetCommand` can generate `BadAlloc` and `BadWindow` errors.

Property

WM_COMMAND

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11),

XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

XSetErrorHandler (3X11)

Name

XSetErrorHandler, XGetErrorText, XDisplayName, XSetIOErrorHandler, XGetErrorDatabaseText – default error handlers

Syntax

XSetErrorHandler(*handler*)
    int (**handler*)(Display *, XErrorEvent *)

XGetErrorText(*display*, *code*, *buffer_return*, *length*)
    Display **display*;
    int *code*;
    char **buffer_return*;
    int *length*;

char *XDisplayName(*string*)
    char **string*;

XSetIOErrorHandler(*handler*)
    int (**handler*)(Display *);

XGetErrorDatabaseText(*display*, *name*, *message*, *default_string*, *buffer_return*, *length*)
    Display **display*;
    char **name*, **message*;
    char **default_string*;
    char **buffer_return*;
    int *length*;

Arguments

buffer_return	Returns the error description.
code	Specifies the error code for which you want to obtain a description.
default_string	Specifies the default error message if none is found in the database.
display	Specifies the connection to the X server.
handler	Specifies the program's supplied error handler.
length	Specifies the size of the buffer.
message	Specifies the type of the error message.
name	Specifies the name of the application.

string Specifies the character string.

Description

Xlib generally calls the program's supplied error handler whenever an error is received. It is not called on `BadName` errors from `OpenFont`, `LookupColor`, or `AllocNamedColor` protocol requests or on `BadFont` errors from a `QueryFont` protocol request. These errors generally are reflected back to the program through the procedural interface. Because this condition is not assumed to be fatal, it is acceptable for your error handler to return. However, the error handler should not call any functions (directly or indirectly) on the display that will generate protocol requests or that will look for input events.

The `XGetErrorText` function copies a null-terminated string describing the specified error code into the specified buffer. It is recommended that you use this function to obtain an error description because extensions to Xlib may define their own error codes and error strings.

The `XDisplayName` function returns the name of the display that `XOpenDisplay` would attempt to use. If a NULL string is specified, `XDisplayName` looks in the environment for the display and returns the display name that `XOpenDisplay` would attempt to use. This makes it easier to report to the user precisely which display the program attempted to open when the initial connection attempt failed.

The `XSetIOErrorHandler` sets the fatal I/O error handler. Xlib calls the program's supplied error handler if any sort of system call error occurs (for example, the connection to the server was lost). This is assumed to be a fatal condition, and the called routine should not return. If the I/O error handler does return, the client process exits.

The `XGetErrorDatabaseText` function returns a message (or the default message) from the error message database. Xlib uses this function internally to look up its error messages. On a UNIX-based system, the error message database is `/usr/lib/X11/XErrorDB`.

The name argument should generally be the name of your application. The message argument should indicate which type of error message you want. Xlib uses three predefined message types to report errors (uppercase and lowercase matter):

XProtoError The protocol error number is used as a string for the message argument.

XlibMessage These are the message strings that are used internally by the library.

XRequest The major request protocol number is used for the message argument. If no string is found in the error database, the default_string is returned to the buffer argument.

See Also

XSynchronize(3X11)
Guide to the Xlib Library

Name

XSetFillStyle, XSetFillRule – GC convenience routines

Syntax

XSetFillStyle(*display*, *gc*, *fill_style*)
Display **display*;
GC *gc*;
int *fill_style*;

XSetFillRule(*display*, *gc*, *fill_rule*)
Display **display*;
GC *gc*;
int *fill_rule*;

Arguments

display	Specifies the connection to the X server.
fill_rule	Specifies the fill-rule you want to set for the specified GC. You can pass `EvenOddRule` or `WindingRule`.
fill_style	Specifies the fill-style you want to set for the specified GC. You can pass `FillSolid`, `FillTiled`, `FillStippled`, or `FillOpaqueStippled`.
gc	Specifies the GC.

Description

The `XSetFillStyle` function sets the fill-style in the specified GC.

`XSetFillStyle` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

The `XSetFillRule` function sets the fill-rule in the specified GC.

`XSetFillRule` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadGC`	A value for a GContext argument does not name a defined GContext.
`BadValue`	Some numeric value falls outside the range of values

accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

Name

XSetFont – GC convenience routines

Syntax

XSetFont(*display*, *gc*, *font*)
Display **display*;
GC *gc*;
Font *font*;

Arguments

display	Specifies the connection to the X server.
font	Specifies the font.
gc	Specifies the GC.

Description

The `XSetFont` function sets the current font in the specified GC.

`XSetFont` can generate `BadAlloc`, `BadFont`, and `BadGC` errors.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadFont`	A value for a Font or GContext argument does not name a defined Font.
`BadGC`	A value for a GContext argument does not name a defined GContext.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetLineAttributes(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

XSetFontPath (3X11)

Name

XSetFontPath, XGetFontPath, XFreeFontPath – set, get, or free the font search path

Syntax

XSetFontPath(*display*, *directories*, *ndirs*)
    Display **display*;
    char ***directories*;
    int *ndirs*;

char **XGetFontPath(*display*, *npaths_return*)
    Display **display*;
    int **npaths_return*;

XFreeFontPath(*list*)
    char ***list*;

Arguments

directories	Specifies the directory path used to look for a font. Setting the path to the empty list restores the default path defined for the X server.
display	Specifies the connection to the X server.
list	Specifies the array of strings you want to free.
ndirs	Specifies the number of directories in the path.
npaths_return	Returns the number of strings in the font path array.

Description

The `XSetFontPath` function defines the directory search path for font lookup. There is only one search path per X server, not one per client. The interpretation of the strings is operating system dependent, but they are intended to specify directories to be searched in the order listed. Also, the contents of these strings are operating system dependent and are not intended to be used by client applications. Usually, the X server is free to cache font information internally rather than having to read fonts from files. In addition, the X server is guaranteed to flush all cached information about fonts for which there currently are no explicit resource IDs allocated. The meaning of an error from this request is operating system dependent.

XSetFontPath can generate a BadValue error.

The XGetFontPath function allocates and returns an array of strings containing the search path. When it is no longer needed, the data in the font path should be freed by using XFreeFontPath.

The XFreeFontPath function frees the data allocated by XGetFontPath.

Diagnostics

BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XListFont(3X11), XLoadFonts(3X11)
Guide to the Xlib Library

XSetIconName (3X11)

Name

XSetIconName, XGetIconName – set or get icon names

Syntax

XSetIconName(*display*, *w*, *icon_name*)
Display **display*;
Window *w*;
char **icon_name*;

Status XGetIconName(*display*, *w*, *icon_name_return*)
Display **display*;
Window *w*;
char ***icon_name_return*;

Arguments

display	Specifies the connection to the X server.
icon_name	Specifies the icon name, which should be a null-terminated string.
icon_name_return	Returns a pointer to the window's icon name, which is a null-terminated string.
w	Specifies the window.

Description

The XSetIconName function sets the name to be displayed in a window's icon.

XSetIconName can generate BadAlloc and BadWindow errors.

The XGetIconName function returns the name to be displayed in the specified window's icon. If it succeeds, it returns nonzero; otherwise, if no icon name has been set for the window, it returns zero. If you never assigned a name to the window, XGetIconName sets icon_name_return to NULL. When finished with it, a client must free the icon name string using XFree.

XGetIconName can generate a BadWindow error.

Property

WM_ICON_NAME

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetIconSizes, XGetIconSizes – set or get icon size hints

Syntax

XSetIconSizes(*display*, *w*, *size_list*, *count*)
Display **display*;
Window *w*;
XIconSize **size_list*;
int *count*;

Status XGetIconSizes(*display*, *w*, *size_list_return*, *count_return*)
Display **display*;
Window *w*;
XIconSize ***size_list_return*;
int **count_return*;

Arguments

display	Specifies the connection to the X server.
count	Specifies the number of items in the size list.
count_return	Returns the number of items in the size list.
size_list	Specifies a pointer to the size list.
size_list_return	Returns a pointer to the size list.
w	Specifies the window.

Description

The `XSetIconSizes` function is used only by window managers to set the supported icon sizes.

`XSetIconSizes` can generate `BadAlloc` and `BadWindow` errors.

The `XGetIconSizes` function returns zero if a window manager has not set icon sizes or nonzero otherwise. `XGetIconSizes` should be called by an application that wants to find out what icon sizes would be most appreciated by the window manager under which the application is running. The application should then use `XSetWMHints` to supply the window manager with an icon pixmap or window in one of the supported sizes. To free the data allocated in size_list_return, use `XFree`.

`XGetIconSizes` can generate a `BadWindow` error.

Property

WM_ICON_SIZE

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconName(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetInputFocus, XGetInputFocus – control input focus

Syntax

XSetInputFocus(*display*, *focus*, *revert_to*, *time*)
Display **display*;
Window *focus*;
int *revert_to*;
Time *time*;

XGetInputFocus(*display*, *focus_return*, *revert_to_return*)
Display **display*;
Window **focus_return*;
int **revert_to_return*;

Arguments

display	Specifies the connection to the X server.
focus	Specifies the window, `PointerRoot`, or `None`.
focus_return	Returns the focus window, `PointerRoot`, or `None`.
revert_to	Specifies where the input focus reverts to if the window becomes not viewable. You can pass `RevertToParent`, `RevertToPointerRoot`, or `RevertToNone`.
revert_to_return	Returns the current focus state (`RevertToParent`, `RevertToPointerRoot`, or `RevertToNone`).
time	Specifies the time. You can pass either a timestamp or `CurrentTime`.

Description

The `XSetInputFocus` function changes the input focus and the last-focus-change time. It has no effect if the specified time is earlier than the current last-focus-change time or is later than the current X server time. Otherwise, the last-focus-change time is set to the specified time (`CurrentTime` is replaced by the current X server time). `XSetInputFocus` causes the X server to generate `FocusIn` and `FocusOut` events.

Depending on the focus argument, the following occurs:

- If focus is `None`, all keyboard events are discarded until a new focus window is set, and the revert_to argument is ignored.
- If focus is a window, it becomes the keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported as usual. Otherwise, the event is reported relative to the focus window.
- If focus is `PointerRoot`, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, the revert_to argument is ignored.

The specified focus window must be viewable at the time `XSetInputFocus` is called, or a `BadMatch` error results. If the focus window later becomes not viewable, the X server evaluates the revert_to argument to determine the new focus window as follows:

- If revert_to is `RevertToParent`, the focus reverts to the parent (or the closest viewable ancestor), and the new revert_to value is taken to be `RevertToNone`.
- If revert_to is `RevertToPointerRoot` or `RevertToNone`, the focus reverts to `PointerRoot` or `None`, respectively. When the focus reverts, the X server generates `FocusIn` and `FocusOut` events, but the last-focus-change time is not affected.

`XSetInputFocus` can generate `BadMatch`, `BadValue`, and `BadWindow` errors.

The `XGetInputFocus` function returns the focus window and the current focus state.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XWarpPointer(3X11)
Guide to the Xlib Library

XSetLineAttribute (3X11)

Name

XSetLineAttribute, XSetDashes – GC convenience routines

Syntax

XSetLineAttributes(*display*, *gc*, *line_width*, *line_style*, *cap_style*, *join_style*)
Display **display*;
GC *gc*;
unsigned int *line_width*;
int *line_style*;
int *cap_style*;
int *join_style*;

XSetDashes(*display*, *gc*, *dash_offset*, *dash_list*, *n*)
Display **display*;
GC *gc*;
int *dash_offset*;
char *dash_list*[];
int *n*;

Arguments

cap_style	Specifies the line-style and cap-style you want to set for the specified GC. You can pass `CapNotLast`, `CapButt`, `CapRound`, or `CapProjecting`.
dash_list	Specifies the dash-list for the dashed line-style you want to set for the specified GC.
dash_offset	Specifies the phase of the pattern for the dashed line-style you want to set for the specified GC.
display	Specifies the connection to the X server.
gc	Specifies the GC.
join_style	Specifies the line join-style you want to set for the specified GC. You can pass `JoinMiter`, `JoinRound`, or `JoinBevel`.
line_style	Specifies the line-style you want to set for the specified GC. You can pass `LineSolid`, `LineOnOffDash`, or `LineDoubleDash`.
line_width	Specifies the line-width you want to set for the specified GC.
n	Specifies the number of elements in dash_list.

Description

The XSetLineAttributes function sets the line drawing components in the specified GC.

XSetLineAttributes can generate BadAlloc, BadGC, and BadValue errors.

The XSetDashes function sets the dash-offset and dash-list attributes for dashed line styles in the specified GC. There must be at least one element in the specified dash_list, or a BadValue error results. The initial and alternating elements (second, fourth, and so on) of the dash_list are the even dashes, and the others are the odd dashes. Each element specifies a dash length in pixels. All of the elements must be nonzero, or a BadValue error results. Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list.

The dash-offset defines the phase of the pattern, specifying how many pixels into the dash-list the pattern should actually begin in any single graphics request. Dashing is continuous through path elements combined with a join-style but is reset to the dash-offset each time a cap-style is applied at a line endpoint.

The unit of measure for dashes is the same for the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but implementations are only required to match this ideal for horizontal and vertical lines. Failing the ideal semantics, it is suggested that the length be measured along the major axis of the line. The major axis is defined as the x axis for lines drawn at an angle of between –45 and +45 degrees or between 315 and 225 degrees from the x axis. For all other lines, the major axis is the y axis.

XSetDashes can generate BadAlloc, BadGC, and BadValue errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadGC	A value for a GContext argument does not name a defined GContext.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of

alternatives can generate this error.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetState(3X11), XSetTile(3X11)
Guide to the Xlib Library

Name

XSetNormalHints, XGetNormalHints – set or get normal state hints

Syntax

XSetNormalHints(*display*, *w*, *hints*)
Display **display*;
Window *w*;
XSizeHints **hints*;

Status XGetNormalHints(*display*, *w*, *hints_return*)
Display **display*;
Window *w*;
XSizeHints **hints_return*;

Arguments

display	Specifies the connection to the X server.
hints	Specifies a pointer to the size hints for the window in its normal state.
hints_return	Returns the size hints for the window in its normal state.
w	Specifies the window.

Description

The `XSetNormalHints` function sets the size hints structure for the specified window. Applications use `XSetNormalHints` to inform the window manager of the size or position desirable for that window. In addition, an application that wants to move or resize itself should call `XSetNormalHints` and specify its new desired location and size as well as making direct Xlib calls to move or resize. This is because window managers may ignore redirected configure requests, but they pay attention to property changes.

To set size hints, an application not only must assign values to the appropriate members in the hints structure but also must set the flags member of the structure to indicate which information is present and where it came from. A call to `XSetNormalHints` is meaningless, unless the flags member is set to indicate which members of the structure have been assigned values.

XSetNormalHints can generate BadAlloc and BadWindow errors.

The XGetNormalHints function returns the size hints for a window in its normal state. It returns a nonzero status if it succeeds or zero if the application specified no normal size hints for this window.

XGetNormalHints can generate a BadWindow error.

Property

WM_NORMAL_HINTS

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetPointerMapping, XGetPointerMapping – manipulate pointer settings

Syntax

int XSetPointerMapping(*display*, *map*, *nmap*)
    Display **display*;
    unsigned char *map*[];
    int *nmap*;

int XGetPointerMapping(*display*, *map_return*, *nmap*)
    Display **display*;
    unsigned char *map_return*[];
    int *nmap*;

Arguments

display	Specifies the connection to the X server.
map	Specifies the mapping list.
map_return	Returns the mapping list.
nmap	Specifies the number of items in the mapping list.

Description

The `XSetPointerMapping` function sets the mapping of the pointer. If it succeeds, the X server generates a `MappingNotify` event, and `XSetPointerMapping` returns `MappingSuccess`. Elements of the list are indexed starting from one. The length of the list must be the same as `XGetPointerMapping` would return, or a `BadValue` error results. The index is a core button number, and the element of the list defines the effective number. A zero element disables a button, and elements are not restricted in value by the number of physical buttons. However, no two elements can have the same nonzero value, or a `BadValue` error results. If any of the buttons to be altered are logically in the down state, `XSetPointerMapping` returns `MappingBusy`, and the mapping is not changed.

`XSetPointerMapping` can generate a `BadValue` error.

The `XGetPointerMapping` function returns the current mapping of the pointer. Elements of the list are indexed starting from one. `XGetPointerMapping` returns the number of physical buttons actually on the pointer. The nominal mapping for a pointer is the identity mapping:

map[i]=i. The nmap argument specifies the length of the array where the pointer mapping is returned, and only the first nmap elements are returned in map_return.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XChangeKeyboardControl(3X11), XChangeKeyboardMapping(3X11)
Guide to the Xlib Library

Name

XSetScreenSaver, XForceScreenSaver, XActivateScreenSaver, XResetScreenSaver, XGetScreenSaver – manipulate the screen saver

Syntax

XSetScreenSaver(*display*, *timeout*, *interval*, *prefer_blanking*, *allow_exposures*)
Display **display*;
int *timeout*, *interval*;
int *prefer_blanking*;
int *allow_exposures*;

XForceScreenSaver(*display*, *mode*)
Display **display*;
int *mode*;

XActivateScreenSaver(*display*)
Display **display*;

XResetScreenSaver(*display*)
Display **display*;

XGetScreenSaver(*display*, *timeout_return*, *interval_return*, *prefer_blanking_return*, *allow_exposures_return*)
Display **display*;
int **timeout_return*, **interval_return*;
int **prefer_blanking_return*;
int **allow_exposures_return*;

Arguments

allow_exposures
: Specifies the screen save control values. You can pass `DontAllowExposures`, `AllowExposures`, or `DefaultExposures`.

allow_exposures_return
: Returns the current screen save control value (`DontAllowExposures`, `AllowExposures`, or `DefaultExposures`).

display
: Specifies the connection to the X server.

interval
: Specifies the interval between screen saver alterations.

interval_return Returns the interval between screen saver invocations.

mode Specifies the mode that is to be applied. You can pass `ScreenSaverActive` or `ScreenSaverReset`.

prefer_blanking
Specifies how to enable screen blanking. You can pass `DontPreferBlanking`, `PreferBlanking`, or `DefaultBlanking`.

prefer_blanking_return
Returns the current screen blanking preference (`DontPreferBlanking`, `PreferBlanking`, or `DefaultBlanking`).

timeout Specifies the timeout, in seconds, until the screen saver turns on.

timeout_return Returns the timeout, in minutes, until the screen saver turns on.

Description

Timeout and interval are specified in seconds. A timeout of 0 disables the screen saver, and a timeout of –1 restores the default. Other negative values generate a `BadValue` error. If the timeout value is nonzero, `XSetScreenSaver` enables the screen saver. An interval of 0 disables the random-pattern motion. If no input from devices (keyboard, mouse, and so on) is generated for the specified number of timeout seconds once the screen saver is enabled, the screen saver is activated.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen simply goes blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending `Expose` events to clients, the screen is tiled with the root window background tile randomly re-origined each interval minutes. Otherwise, the screens' state do not change, and the screen saver is not activated. The screen saver is deactivated, and all screen states are restored at the next keyboard or pointer input or at the next call to `XForceScreenSaver` with mode `ScreenSaverReset`.

If the server-dependent screen saver method supports periodic change, the interval argument serves as a hint about how long the change period should be, and zero hints that no periodic change should be made. Examples of ways to change the screen include scrambling the colormap periodically, moving an icon image around the screen periodically, or tiling the screen with the root window background tile, randomly re-origined periodically.

XSetScreenSaver can generate a BadValue error.

If the specified mode is ScreenSaverActive and the screen saver currently is deactivated, XForceScreenSaver activates the screen saver even if the screen saver had been disabled with a timeout of zero. If the specified mode is ScreenSaverReset and the screen saver currently is enabled, XForceScreenSaver deactivates the screen saver if it was activated, and the activation timer is reset to its initial state (as if device input had been received).

XForceScreenSaver can generate a BadValue error.

The XActivateScreenSaver function activates the screen saver.

The XResetScreenSaver function resets the screen saver.

The XGetScreenSaver function gets the current screen saver values.

Diagnostics

`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

Guide to the Xlib Library

Name

XSetSelectionOwner, XGetSelectionOwner, XConvertSelection – manipulate window selection

Syntax

XSetSelectionOwner(*display*, *selection*, *owner*, *time*)
Display **display*;
Atom *selection*;
Window *owner*;
Time *time*;

Window XGetSelectionOwner(*display*, *selection*)
Display **display*;
Atom *selection*;

XConvertSelection(*display*, *selection*, *target*, *property*, *requestor*, *time*)
Display **display*;
Atom *selection*, *target*;
Atom *property*;
Window *requestor*;
Time *time*;

Arguments

display	Specifies the connection to the X server.
owner	Specifies the owner of the specified selection atom. You can pass a window or `None`.
property	Specifies the property name. You also can pass `None`.
requestor	Specifies the requestor.
selection	Specifies the selection atom.
target	Specifies the target atom.
time	Specifies the time. You can pass either a timestamp or `CurrentTime`.

Description

The `XSetSelectionOwner` function changes the owner and last-change time for the specified selection and has no effect if the specified time is earlier than the current last-change time of the specified selection or is later than the current X server time. Otherwise, the last-change time is set to the

specified time, with CurrentTime replaced by the current server time. If the owner window is specified as None, then the owner of the selection becomes None (that is, no owner). Otherwise, the owner of the selection becomes the client executing the request.

If the new owner (whether a client or None) is not the same as the current owner of the selection and the current owner is not None, the current owner is sent a SelectionClear event. If the client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, the owner of the selection automatically reverts to None, but the last-change time is not affected. The selection atom is uninterpreted by the X server. XGetSelectionOwner returns the owner window, which is reported in SelectionRequest and SelectionClear events. Selections are global to the X server.

XSetSelectionOwner can generate BadAtom and BadWindow errors.

The XGetSelectionOwner function returns the window ID associated with the window that currently owns the specified selection. If no selection was specified, the function returns the constant None. If None is returned, there is no owner for the selection.

XGetSelectionOwner can generate a BadAtom error.

XConvertSelection requests that the specified selection be converted to the specified target type:

- If the specified selection has an owner, the X server sends a SelectionRequest event to that owner.
- If no owner for the specified selection exists, the X server generates a SelectionNotify event to the requestor with property None.

In either event, the arguments are passed on unchanged. There are two predefined selection atoms: PRIMARY and SECONDARY.

XConvertSelection can generate BadAtom and BadWindow errors.

Diagnostics

BadAtom	A value for an Atom argument does not name a defined Atom.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

Guide to the Xlib Library

XSetSizeHints (3X11)

Name

XSetSizeHints, XGetSizeHints – set or get window size hints

Syntax

XSetSizeHints(*display*, *w*, *hints*, *property*)
Display **display*;
Window *w*;
XSizeHints **hints*;
Atom *property*;

Status XGetSizeHints(*display*, *w*, *hints_return*, *property*)
Display **display*;
Window *w*;
XSizeHints **hints_return*;
Atom *property*;

Arguments

display	Specifies the connection to the X server.
hints	Specifies a pointer to the size hints.
hints_return	Returns the size hints.
property	Specifies the property name.
w	Specifies the window.

Description

The `XSetSizeHints` function sets the `XSizeHints` structure for the named property and the specified window. This is used by `XSetNormalHints` and `XSetZoomHints`, and can be used to set the value of any property of type WM_SIZE_HINTS. Thus, it may be useful if other properties of that type get defined.

`XSetSizeHints` can generate `BadAlloc`, `BadAtom`, and `BadWindow` errors.

`XGetSizeHints` returns the `XSizeHints` structure for the named property and the specified window. This is used by `XGetNormalHints` and `XGetZoomHints`. It also can be used to retrieve the value of any property of type WM_SIZE_HINTS. Thus, it may be useful if other properties of that type get defined. `XGetSizeHints` returns a nonzero status if a size hint was defined or zero otherwise.

XGetSizeHints can generate BadAtom and BadWindow errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

XSetStandardColormap (3X11)

Name

XSetStandardColormap, XGetStandardColormap – set or get standard colormaps

Syntax

XSetStandardColormap(*display*, *w*, *colormap*, *property*)
Display **display*;
Window *w*;
XStandardColormap **colormap*;
Atom *property*; /* RGB_BEST_MAP, etc. */

Status XGetStandardColormap(*display*, *w*, *colormap_return*, *property*)
Display **display*;
Window *w*;
XStandardColormap **colormap_return*;
Atom *property*; /* RGB_BEST_MAP, etc. */

Arguments

colormap	Specifies the colormap.
colormap_return	Returns the colormap associated with the specified atom.
display	Specifies the connection to the X server.
property	Specifies the property name.
w	Specifies the window.

Description

The `XSetStandardColormap` function usually is only used by window managers. To create a standard colormap, follow this procedure:

1. Open a new connection to the same server.
2. Grab the server.
3. See if the property is on the property list of the root window for the screen.
4. If the desired property is not present:
   - Create a colormap (not required for RGB_DEFAULT_MAP)
   - Determine the color capabilities of the display.

- Call XAllocColorPlanes or XAllocColorCells to allocate cells in the colormap.
- Call XStoreColors to store appropriate color values in the colormap.
- Fill in the descriptive members in the XStandardColormap structure.
- Attach the property to the root window.
- Use XSetCloseDownMode to make the resource permanent.

5. Ungrab the server.

XSetStandardColormap can generate BadAlloc, BadAtom, and BadWindow errors.

The XGetStandardColormap function returns the colormap definition associated with the atom supplied as the property argument. For example, to fetch the standard GrayScale colormap for a display, you use XGetStandardColormap with the following syntax:

XGetStandardColormap(dpy, DefaultRootWindow(dpy), &cmap, XA_RGB_GRAY_MAP

Once you have fetched a standard colormap, you can use it to convert RGB values into pixel values. For example, given an XStandardColormap structure and floating-point RGB coefficients in the range 0.0 to 1.0, you can compose pixel values with the following C expression:

```
pixel = base_pixel
        + ((unsigned long) (0.5 + r * red_max)) * red_mult
        + ((unsigned long) (0.5 + g * green_max)) * green_mult
        + ((unsigned long) (0.5 + b * blue_max)) * blue_mult;
```

The use of addition rather than logical OR for composing pixel values permits allocations where the RGB value is not aligned to bit boundaries.

XGetStandardColormap can generate BadAtom and BadWindow errors.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

Guide to the Xlib Library

Name

XSetStandardProperties – set standard window manager properties

Syntax

XSetStandardProperties(*display*, *w*, *window_name*, *icon_name*, *icon_pixmap*, *argv*, *argc*, *hints*)

Display **display*;
Window *w*;
char **window_name*;
char **icon_name*;
Pixmap *icon_pixmap*;
char ***argv*;
int *argc*;
XSizeHints **hints*;

Arguments

argc	Specifies the number of arguments.
argv	Specifies the application's argument list.
display	Specifies the connection to the X server.
hints	Specifies a pointer to the size hints for the window in its normal state.
icon_name	Specifies the icon name, which should be a null-terminated string.
icon_pixmap	Specifies the bitmap that is to be used for the icon or `None`.
w	Specifies the window.
window_name	Specifies the window name, which should be a null-terminated string.

Description

The `XSetStandardProperties` function provides a means by which simple applications set the most essential properties with a single call. `XSetStandardProperties` should be used to give a window manager some information about your program's preferences. It should not be used by applications that need to communicate more information than is possible with `XSetStandardProperties`. (Typically, argv is the argv array of your main program.)

`XSetStandardProperties` can generate `BadAlloc` and `BadWindow` errors.

Properties

WM_NAME, WM_ICON_NAME, WM_HINTS, WM_COMMAND, and WM_NORMALHINTS

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetTransientForHint(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetState, XSetFunction, XSetPlaneMask, XSetForeground, XSetBackground – GC convenience routines

Syntax

XSetState(*display*, *gc*, *foreground*, *background*, *function*, *plane_mask*)
Display **display*;
GC *gc*;
unsigned long *foreground*, *background*;
int *function*;
unsigned long *plane_mask*;

XSetFunction(*display*, *gc*, *function*)
Display **display*;
GC *gc*;
int *function*;

XSetPlaneMask(*display*, *gc*, *plane_mask*)
Display **display*;
GC *gc*;
unsigned long *plane_mask*;

XSetForeground(*display*, *gc*, *foreground*)
Display **display*;
GC *gc*;
unsigned long *foreground*;

XSetBackground(*display*, *gc*, *background*)
Display **display*;
GC *gc*;
unsigned long *background*;

Arguments

background	Specifies the background you want to set for the specified GC.
display	Specifies the connection to the X server.
foreground	Specifies the foreground you want to set for the specified GC.
function	Specifies the function you want to set for the specified GC.
gc	Specifies the GC.

plane_mask Specifies the plane mask.

Description

The `XSetState` function sets the foreground, background, plane mask, and function components for the specified GC.

`XSetState` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

`XSetFunction` sets a specified value in the specified GC.

`XSetFunction` can generate `BadAlloc`, `BadGC`, and `BadValue` errors.

The `XSetPlaneMask` function sets the plane mask in the specified GC.

`XSetPlaneMask` can generate `BadAlloc` and `BadGC` errors.

The `XSetForeground` function sets the foreground in the specified GC.

`XSetForeground` can generate `BadAlloc` and `BadGC` errors.

The `XSetBackground` function sets the background in the specified GC.

`XSetBackground` can generate `BadAlloc` and `BadGC` errors.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadGC`	A value for a GContext argument does not name a defined GContext.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetTile(3X11)
Guide to the Xlib Library

Name

XSetTile, XSetStipple, XSetTSOrigin – GC convenience routines

Syntax

XSetTile(*display*, *gc*, *tile*)
Display **display*;
GC *gc*;
Pixmap *tile*;

XSetStipple(*display*, *gc*, *stipple*)
Display **display*;
GC *gc*;
Pixmap *stipple*;

XSetTSOrigin(*display*, *gc*, *ts_x_origin*, *ts_y_origin*)
Display **display*;
GC *gc*;
int *ts_x_origin*, *ts_y_origin*;

Arguments

display	Specifies the connection to the X server.
gc	Specifies the GC.
stipple	Specifies the stipple you want to set for the specified GC.
tile	Specifies the fill tile you want to set for the specified GC.
ts_x_origin *ts_y_origin*	Specify the x and y coordinates of the tile and stipple origin.

Description

The `XSetTile` function sets the fill tile in the specified GC. The tile and GC must have the same depth, or a `BadMatch` error results.

`XSetTile` can generate `BadAlloc`, `BadGC`, `BadMatch`, and `BadPixmap` errors.

The `XSetStipple` function sets the stipple in the specified GC. The stipple and GC must have the same depth, or a `BadMatch` error results.

`XSetStipple` can generate `BadAlloc`, `BadGC`, `BadMatch`, and `BadPixmap` errors.

XSetTile (3X11)

The `XSetTSOrigin` function sets the tile/stipple origin in the specified GC. When graphics requests call for tiling or stippling, the parent's origin will be interpreted relative to whatever destination drawable is specified in the graphics request.

`XSetTSOrigin` can generate `BadAlloc` and `BadGC` errors.

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadGC`	A value for a GContext argument does not name a defined GContext.
`BadMatch`	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
`BadPixmap`	A value for a Pixmap argument does not name a defined Pixmap.

See Also

XCreateGC(3X11), XQueryBestSize(3X11), XSetArcMode(3X11), XSetClipOrigin(3X11), XSetFillStyle(3X11), XSetFont(3X11), XSetLineAttributes(3X11), XSetState(3X11)
Guide to the Xlib Library

Name

XSetTransientForHint, XGetTransientForHint – set or get transient for hint

Syntax

XSetTransientForHint(*display*, *w*, *prop_window*)
Display **display*;
Window *w*;
Window *prop_window*;

Status XGetTransientForHint(*display*, *w*, *prop_window_return*)
Display **display*;
Window *w*;
Window **prop_window_return*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.
prop_window	Specifies the window that the WM_TRANSIENT_FOR property is to be set to.
prop_window_return	Returns the WM_TRANSIENT_FOR property of the specified window.

Description

The `XSetTransientForHint` function sets the WM_TRANSIENT_FOR property of the specified window to the specified prop_window.

`XSetTransientForHint` can generate `BadAlloc` and `BadWindow` errors.

The `XGetTransientForHint` function returns the WM_TRANSIENT_FOR property for the specified window.

`XGetTransientForHint` can generate a `BadWindow` error.

Property

WM_TRANSIENT_FOR

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetWMHints(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetWMHints, XGetWMHints – set or get window manager hints

Syntax

XSetWMHints(*display*, *w*, *wmhints*)
Display **display*;
Window *w*;
XWMHints **wmhints*;

XWMHints *XGetWMHints(*display*, *w*)
Display **display*;
Window *w*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.
wmhints	Specifies a pointer to the window manager hints.

Description

The `XSetWMHints` function sets the window manager hints that include icon information and location, the initial state of the window, and whether the application relies on the window manager to get keyboard input.

`XSetWMHints` can generate `BadAlloc` and `BadWindow` errors.

The `XGetWMHints` function reads the window manager hints and returns NULL if no WM_HINTS property was set on the window or a pointer to a `XWMHints` structure if it succeeds. When finished with the data, free the space used for it by calling `XFree`.

`XGetWMHints` can generate a `BadWindow` error.

Property

WM_HINTS

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11), XSetZoomHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XSetZoomHints, XGetZoomHints – set or get zoom state hints

Syntax

XSetZoomHints(*display*, *w*, *zhints*)
Display **display*;
Window *w*;
XSizeHints **zhints*;

Status XGetZoomHints(*display*, *w*, *zhints_return*)
Display **display*;
Window *w*;
XSizeHints **zhints_return*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.
zhints	Specifies a pointer to the zoom hints.
zhints_return	Returns the zoom hints.

Description

Many window managers think of windows in one of three states: iconic, normal, or zoomed. The `XSetZoomHints` function provides the window manager with information for the window in the zoomed state.

`XSetZoomHints` can generate `BadAlloc` and `BadWindow` errors.

The `XGetZoomHints` function returns the size hints for a window in its zoomed state. It returns a nonzero status if it succeeds or zero if the application specified no zoom size hints for this window.

`XGetZoomHints` can generate a `BadWindow` error.

Property

WM_ZOOM_HINTS

Diagnostics

`BadAlloc`	The server failed to allocate the requested resource or server memory.
`BadWindow`	A value for a Window argument does not name a defined Window.

See Also

XSetClassHint(3X11), XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetTransientForHint(3X11), XSetWMHints(3X11), XStoreName(3X11)
Guide to the Xlib Library

Name

XStartStat, XStopStat, XPrintStat – start, stop, or display process statistics

Syntax

XStartStat(*display*)
    Display **display*;

XStopStat(*display*)
    Display **display*;

XPrintStat(*display*, *file*)
    Display **display*;
    FILE *file*;

Arguments

display	Specifies the connection to the X server.
file	Specifies the file to which the statistics are to be written.

Description

The `XStartStat` function turns on client-side statistics gathering mode.

The `XStartStop` function turns off client-side statistics gathering mode.

The `XPrintStat` function write the client-side statistics to the specified file. If file is not stdout or stderr, you must open the file before `XPrintStat` can write the statistics to it.

See Also

Guide to the Xlib Library

XStoreBytes (3X11)

Name

XStoreBytes, XStoreBuffer, XFetchBytes, XFetchBuffer, XRotateBuffers – manipulate cut and paste buffers

Syntax

XStoreBytes(*display*, *bytes*, *nbytes*)
Display **display*;
char **bytes*;
int *nbytes*;

XStoreBuffer(*display*, *bytes*, *nbytes*, *buffer*)
Display **display*;
char **bytes*;
int *nbytes*;
int *buffer*;

char *XFetchBytes(*display*, *nbytes_return*)
Display **display*;
int **nbytes_return*;

char *XFetchBuffer(*display*, *nbytes_return*, *buffer*)
Display **display*;
int **nbytes_return*;
int *buffer*;

XRotateBuffers(*display*, *rotate*)
Display **display*;
int *rotate*;

Arguments

buffer	Specifies the buffer in which you want to store the bytes or from which you want the stored data returned.
bytes	Specifies the bytes, which are not necessarily ASCII or null-terminated.
display	Specifies the connection to the X server.
nbytes	Specifies the number of bytes to be stored.
nbytes_return	Returns the number of bytes in the buffer.
rotate	Specifies how much to rotate the cut buffers.

Description

Note that the cut buffer's contents need not be text, so zero bytes are not special. The cut buffer's contents can be retrieved later by any client calling XFetchBytes.

XStoreBytes can generate a BadAlloc error.

If the property for the buffer has never been created, a BadAtom error results.

XStoreBuffer can generate BadAlloc and BadAtom errors.

The XFetchBytes function returns the number of bytes in the nbytes_return argument, if the buffer contains data. Otherwise, the function returns NULL and sets nbytes to 0. The appropriate amount of storage is allocated and the pointer returned. The client must free this storage when finished with it by calling XFree. Note that the cut buffer does not necessarily contain text, so it may contain embedded zero bytes and may not terminate with a null byte.

The XFetchBuffer function returns zero to the nbytes_return argument if there is no data in the buffer.

XFetchBuffer can generate a BadValue error.

The XRotateBuffers function rotates the cut buffers, such that buffer 0 becomes buffer n, buffer 1 becomes n + 1 mod 8, and so on. This cut buffer numbering is global to the display. Note that XRotateBuffers generates BadMatch errors if any of the eight buffers have not been created.

XRotateBuffers can generate a BadMatch error.

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of

alternatives can generate this error.

See Also

Guide to the Xlib Library

Name

XStoreColors, XStoreColor, XStoreNamedColor – set colors

Syntax

XStoreColors(*display*, *colormap*, *color*, *ncolors*)
Display **display*;
Colormap *colormap*;
XColor *color*[];
int *ncolors*;

XStoreColor(*display*, *colormap*, *color*)
Display **display*;
Colormap *colormap*;
XColor **color*;

XStoreNamedColor(*display*, *colormap*, *color*, *pixel*, *flags*)
Display **display*;
Colormap *colormap*;
char **color*;
unsigned long *pixel*;
int *flags*;

Arguments

color	Specifies the pixel and RGB values or the color name string (for example, red).
color	Specifies an array of color definition structures to be stored.
colormap	Specifies the colormap.
display	Specifies the connection to the X server.
flags	Specifies which red, green, and blue components are set.
ncolors	Specifies the number of `XColor` structures in the color definition array.
pixel	Specifies the entry in the colormap.

Description

The `XStoreColors` function changes the colormap entries of the pixel values specified in the pixel members of the `XColor` structures. You specify which color components are to be changed by setting `DoRed`, `DoGreen`, and/or `DoBlue` in the flags member of the `XColor` structures.

If the colormap is an installed map for its screen, the changes are visible immediately. XStoreColors changes the specified pixels if they are allocated writable in the colormap by any client, even if one or more pixels generates an error. If a specified pixel is not a valid index into the colormap, a BadValue error results. If a specified pixel either is unallocated or is allocated read-only, a BadAccess error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XStoreColors can generate BadAccess, BadColor, and BadValue errors.

The XStoreColor function changes the colormap entry of the pixel value specified in the pixel member of the XColor structure. You specified this value in the pixel member of the XColor structure. This pixel value must be a read/write cell and a valid index into the colormap. If a specified pixel is not a valid index into the colormap, a BadValue error results. XStoreColor also changes the red, green, and/or blue color components. You specify which color components are to be changed by setting DoRed, DoGreen, and/or DoBlue in the flags member of the XColor structure. If the colormap is an installed map for its screen, the changes are visible immediately.

XStoreColor can generate BadAccess, BadColor, and BadValue errors.

The XStoreNamedColor function looks up the named color with respect to the screen associated with the colormap and stores the result in the specified colormap. The pixel argument determines the entry in the colormap. The flags argument determines which of the red, green, and blue components are set. You can set this member to the bitwise inclusive OR of the bits DoRed, DoGreen, and DoBlue. If the specified pixel is not a valid index into the colormap, a BadValue error results. If the specified pixel either is unallocated or is allocated read-only, a BadAccess error results. You should use the ISO Latin-1 encoding; uppercase and lowercase do not matter.

XStoreNamedColor can generate BadAccess, BadColor, BadName, and BadValue errors.

Diagnostics

BadAccess	A client attempted to free a color map entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only color map entry.

`BadColor`	A value for a Colormap argument does not name a defined Colormap.
`BadName`	A font or color of the specified name does not exist.
`BadValue`	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

See Also

XAllocColor(3X11), XCreateColormap(3X11), XQueryColor(3X11)
Guide to the Xlib Library

XStoreName (3X11)

Name

XStoreName, XFetchName – set or get window names

Syntax

XStoreName(*display*, *w*, *window_name*)
Display **display*;
Window *w*;
char **window_name*;

Status XFetchName(*display*, *w*, *window_name_return*)
Display **display*;
Window *w*;
char ***window_name_return*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.
window_name	Specifies the window name, which should be a null-terminated string.
window_name_return	Returns a pointer to the window name, which is a null-terminated string.

Description

The `XStoreName` function assigns the name passed to window_name to the specified window. A window manager can display the window name in some prominent place, such as the title bar, to allow users to identify windows easily. Some window managers may display a window's name in the window's icon, although they are encouraged to use the window's icon name if one is provided by the application.

`XStoreName` can generate `BadAlloc` and `BadWindow` errors.

The `XFetchName` function returns the name of the specified window. If it succeeds, it returns nonzero; otherwise, if no name has been set for the window, it returns zero. If the WM_NAME property has not been set for this window, `XFetchName` sets window_name_return to NULL. When finished with it, a client must free the window name string using `XFree`.

XFetchName can generate a BadWindow error.

Property

WM_NAME

Diagnostics

BadAlloc	The server failed to allocate the requested resource or server memory.
BadWindow	A value for a Window argument does not name a defined Window.

See Also

XSetCommand(3X11), XSetIconName(3X11), XSetIconSizeHints(3X11), XSetNormalHints(3X11), XSetSizeHints(3X11), XSetStandardProperties(3X11), XSetWMHints(3X11), XSetZoomHints(3X11)
Guide to the Xlib Library

XStringToKeysym (3X11)

Name

XStringToKeysym, XKeysymToString, XKeycodeToKeysym, XKeysymToKeycode – convert keysyms

Syntax

KeySym XStringToKeysym(*string*)
    char **string*;

char *XKeysymToString(*keysym*)
    KeySym *keysym*;

KeySym XKeycodeToKeysym(*display*, *keycode*, *index*)
    Display **display*;
    KeyCode *keycode*;
    int *index*;

KeyCode XKeysymToKeycode(*display*, *keysym*)
    Display **display*;
    KeySym *keysym*;

Arguments

display	Specifies the connection to the X server.
index	Specifies the element of KeyCode vector.
keycode	Specifies the KeyCode.
keysym	Specifies the KeySym that is to be searched for or converted.
string	Specifies the name of the KeySym that is to be converted.

Description

Valid KeySym names are listed in <X11/keysymdef.h> by removing the XK_ prefix from each name. If the specified string does not match a valid KeySym, XStringToKeysym returns NoSymbol.

The returned string is in a static area and must not be modified. If the specified KeySym is not defined, XKeysymToString returns a NULL.

The XKeycodeToKeysym function uses internal Xlib tables and returns the KeySym defined for the specified KeyCode and the element of the KeyCode vector. If no symbol is defined, XKeycodeToKeysym returns NoSymbol.

If the specified KeySym is not defined for any KeyCode, `XKeysymToKeycode` returns zero.

See Also

XLookupKeysym(3X11)
Guide to the Xlib Library

XSynchronize (3X11)

Name

XSynchronize, XSetAfterFunction – enable or disable synchronization

Syntax

int (*XSynchronize(*display*, *onoff*))()
    Display **display*;
    Bool *onoff*;

int (*XSetAfterFunction(*display*, *procedure*))()
    Display **display*;
    int (**procedure*)();

Arguments

display	Specifies the connection to the X server.
procedure	Specifies the function to be called after an Xlib function that generates a protocol request completes its work.
onoff	Specifies a Boolean value that indicates whether to enable or disable synchronization.

Description

The `XSynchronize` function returns the previous after function. If onoff is `True`, `XSynchronize` turns on synchronous behavior. If onoff is `False`, `XSynchronize` turns off synchronous behavior.

The specified procedure is called with only a display pointer. `XSetAfterFunction` returns the previous after function.

See Also

XSetErrorHandler(3X11)
Guide to the Xlib Library

Name

XTextExtents, XTextExtents16, XQueryTextExtents, XQueryTextExtents16 – compute or query text extents

Syntax

XTextExtents(*font_struct*, *string*, *nchars*, *direction_return*, *font_ascent_return*, *font_descent_return*, *overall_return*)
XFontStruct **font_struct*;
char **string*;
int *nchars*;
int **direction_return*;
int **font_ascent_return*, **font_descent_return*;
XCharStruct **overall_return*;

XTextExtents16(*font_struct*, *string*, *nchars*, *direction_return*, *font_ascent_return*, *font_descent_return*, *overall_return*)
XFontStruct **font_struct*;
XChar2b **string*;
int *nchars*;
int **direction_return*;
int **font_ascent_return*, **font_descent_return*;
XCharStruct **overall_return*;

XQueryTextExtents(*display*, *font_ID*, *string*, *nchars*, *direction_return*, *font_ascent_return*, *font_descent_return*, *overall_return*)
Display **display*;
XID *font_ID*;
char **string*;
int *nchars*;
int **direction_return*;
int **font_ascent_return*, **font_descent_return*;
XCharStruct **overall_return*;

XQueryTextExtents16(*display*, *font_ID*, *string*, *nchars*, *direction_return*, *font_ascent_return*, *font_descent_return*, *overall_return*)
Display **display*;
XID *font_ID*;
XChar2b **string*;
int *nchars*;
int **direction_return*;

int **font_ascent_return*, **font_descent_return*;
XCharStruct **overall_return*;

Arguments

direction_return
: Returns the value of the direction hint (`FontLeftToRight` or `FontRightToLeft`).

display
: Specifies the connection to the X server.

font_ID
: Specifies either the font ID or the `GContext` ID that contains the font.

font_ascent_return
: Returns the font ascent.

font_descent_return
: Returns the font descent.

font_struct
: Specifies a pointer to the `XFontStruct` structure.

nchars
: Specifies the number of characters in the character string.

string
: Specifies the character string.

overall_return
: Returns the overall size in the specified `XCharStruct` structure.

Description

The `XTextExtents` and `XTextExtents16` functions perform the size computation locally and, thereby, avoid the round-trip overhead of `XQueryTextExtents` and `XQueryTextExtents16`. Both functions return an `XCharStruct` structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbearing member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each XChar2b structure is interpreted as a 16-bit number with byte1 as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

The XQueryTextExtents and XQueryTextExtents16 functions return the bounding box of the specified 8-bit and 16-bit character string in the specified font or the font contained in the specified GC. These functions query the X server and, therefore, suffer the round-trip overhead that is avoided by XTextExtents and XTextExtents16. Both functions return a XCharStruct structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbearing member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each XChar2b structure is interpreted as a 16-bit number with byte1 as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

XQueryTextExtents XQueryTextExtents16 and can generate BadFont and BadGC errors.

Diagnostics

BadFont	A value for a Font or GContext argument does not name a defined Font.
BadGC	A value for a GContext argument does not name a defined GContext.

See Also

XTextWidth(3X11)
Guide to the Xlib Library

XTextWidth(3X11)

Name

XTextWidth, XTextWidth16 – compute text width

Syntax

int XTextWidth(*font_struct*, *string*, *count*)
    XFontStruct **font_struct*;
    char **string*;
    int *count*;

int XTextWidth16(*font_struct*, *string*, *count*)
    XFontStruct **font_struct*;
    XChar2b **string*;
    int *count*;

Arguments

count	Specifies the character count in the specified string.
font_struct	Specifies the font used for the width computation.
string	Specifies the character string.

Description

The `XTextWidth` and `XTextWidth16` functions return the width of the specified 8-bit or 2-byte character strings.

See Also

XTextExtents(3X11)
Guide to the Xlib Library

Name

XTranslateCoordinates – translate window coordinates

Syntax

Bool XTranslateCoordinates(*display*, *src_w*, *dest_w*, *src_x*, *src_y*, *dest_x_return*, *dest_y_return*, *child_return*)
Display **display*;
Window *src_w*, *dest_w*;
int *src_x*, *src_y*;
int **dest_x_return*, **dest_y_return*;
Window **child_return*;

Arguments

child_return	Returns the child if the coordinates are contained in a mapped child of the destination window.
dest_w	Specifies the destination window.
dest_x_return *dest_y_return*	Return the x and y coordinates within the destination window.
display	Specifies the connection to the X server.
src_w	Specifies the source window.
src_x *src_y*	Specify the x and y coordinates within the source window.

Description

The `XTranslateCoordinates` function takes the src_x and src_y coordinates relative to the source window's origin and returns these coordinates to dest_x_return and dest_y_return relative to the destination window's origin. If `XTranslateCoordinates` returns zero, src_w and dest_w are on different screens, and dest_x_return and dest_y_return are zero. If the coordinates are contained in a mapped child of dest_w, that child is returned to child_return. Otherwise, child_return is set to `None`.

`XTranslateCoordinates` can generate a `BadWindow` error.

Diagnostics

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

Guide to the Xlib Library

Name

XUnmapWindow, XUnmapSubwindows – unmap windows

Syntax

XUnmapWindow(*display*, *w*)
    Display **display*;
    Window *w*;

XUnmapSubwindows(*display*, *w*)
    Display **display*;
    Window *w*;

Arguments

display	Specifies the connection to the X server.
w	Specifies the window.

Description

The `XUnmapWindow` function unmaps the specified window and causes the X server to generate an `UnmapNotify` event. If the specified window is already unmapped, `XUnmapWindow` has no effect. Normal exposure processing on formerly obscured windows is performed. Any child window will no longer be visible until another map call is made on the parent. In other words, the subwindows are still mapped but are not visible until the parent is mapped. Unmapping a window will generate `Expose` events on windows that were formerly obscured by it.

`XUnmapWindow` can generate a `BadWindow` error.

The `XUnmapSubwindows` function unmaps all subwindows for the specified window in bottom-to-top stacking order. It causes the X server to generate an `UnmapNotify` event on each subwindow and `Expose` events on formerly obscured windows. Using this function is much more efficient than unmapping multiple windows one at a time because the server needs to perform much of the work only once, for all of the windows, rather than for each window.

`XUnmapSubwindows` can generate a `BadWindow` error.

Diagnostics

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

XChangeWindowAttributes(3X11), XConfigureWindow(3X11), XCreateWindow(3X11), XDestroyWindow(3X11), XMapWindow(3X11) XRaiseWindow(3X11)
Guide to the Xlib Library

Name

XWarpPointer – move pointer

Syntax

XWarpPointer(*display*, *src_w*, *dest_w*, *src_x*, *src_y*, *src_width*, *src_height*, *dest_x*, *dest_y*)

Display **display*;
Window *src_w*, *dest_w*;
int *src_x*, *src_y*;
unsigned int *src_width*, *src_height*;
int *dest_x*, *dest_y*;

Arguments

dest_w	Specifies the destination window or `None`.
dest_x *dest_y*	Specify the x and y coordinates within the destination window.
display	Specifies the connection to the X server.
src_x *src_y* *src_width* *src_height*	Specify a rectangle in the source window.
src_w	Specifies the source window or `None`.

Description

If dest_w is `None`, `XWarpPointer` moves the pointer by the offsets (dest_x, dest_y) relative to the current position of the pointer. If dest_w is a window, `XWarpPointer` moves the pointer to the offsets (dest_x, dest_y) relative to the origin of dest_w. However, if src_w is a window, the move only takes place if the specified rectangle src_w contains the pointer.

The src_x and src_y coordinates are relative to the origin of src_w. If src_height is zero, it is replaced with the current height of src_w minus src_y. If src_width is zero, it is replaced with the current width of src_w minus src_x.

There is seldom any reason for calling this function. The pointer should normally be left to the user. If you do use this function, however, it generates events just as if the user had instantaneously moved the pointer

from one position to another. Note that you cannot use `XWarpPointer` to move the pointer outside the confine_to window of an active pointer grab. An attempt to do so will only move the pointer as far as the closest edge of the confine_to window.

`XWarpPointer` can generate a `BadWindow` error.

Diagnostics

`BadWindow` A value for a Window argument does not name a defined Window.

See Also

XSetInputFocus(3X11)
Guide to the Xlib Library

Insert tabbed
divider here.
Then discard
this sheet.

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.

See Also

XtCallCallbacks(3Xt)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAddGrab (3Xt)

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppAddActions(3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

Name

XtAppAddConverter – register a 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 the specified resource converter.

See Also

XtConvert(3Xt), XtStringConversionWarning(3Xt)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppAddInput(3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppAddTimeOut(3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppCreateShell (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppError (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppErrorMsg (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppGetErrorDatabase (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtAppGetSelectionTimeout (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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 XUI 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 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 XUI 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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtBuildEventMask (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtCallCallbacks (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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 is 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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

Name

XtCreatePopupShell – create a pop-up 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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtCreateWidget (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtCreateWindow (3Xt)

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 other geometry information.
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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtDisplay (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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 from which to get the resources. 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 *Guide to the Xlib Library*.

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtGetGC (3Xt)

Name

XtGetGC, XtReleaseGC – obtain and destroy a shareable 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 shareable, 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 *Guide to the Xlib Library*.

The `XtReleaseGC` function deallocates the specified shared GC.

See Also

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

Name

XtGetResourceList – obtain a 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 pointer for the created widget.

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtGetSelectionValue (3Xt)

Name

XtGetSelectionValue, XtGetSelectionValues, XtGetSelectionValueIncremental, XtGetSelectionValuesIncremental – 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*;

void XtGetSelectionValueIncremental(*w*, *selection*, *target*, *selection_callback*, *cancel_callback*, *client_data*, *time*)
Widget *w*;
Atom *selection*;
Atom *target*;
XtSelectionIncrCallbackProc *selection_callback*;
XtCancelSelectionCallbackProc *cancel_callback*;
caddr_t *client_data*;
Time *time*;

void XtGetSelectionValuesIncremental(*w*, *selection*, *targets*, *count*, *selection_callback*, *cancel_callback*, *client_data*, *time*)
Widget *w*;
Atom *selection*;
Atom **targets*;
int *count*;
XtSelectionIncrCallbackProc *selection_callback*;
XtCancelConvertSelectionProc *cancel_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.

cancel_callback
Specifies the callback procedure that is to be called if the connection is lost.

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).

selection_callback
Specifies the callback procedure that is to be called to obtain the next incremental chunk of data or for each selection value obtained.

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.

The XtGetSelectionValueIncremental function is similar to XtGetSelectionValue except that the callback procedure will be called repeatedly each time upon delivery of the next segment of the selection value. The end of the selection value is detected when callback is called with a value of length zero. If the transfer of the selection is aborted in the middle of a transfer, the cancel_callback procedure is called so that the requestor can dispose of the partial selection value it has collected up until that point.

The XtGetSelectionValuesIncremental function is similar to XtGetSelectionValueIncremental except that it takes a list of targets and client_data. XtGetSelectionValuesIncremental is equivalent to calling XtGetSelectionValueIncremental successively for each target/client_data pair. XtGetSelectionValuesIncremental does guarantee that all the conversions will use the same selection value because the ownership of the selection cannot change in the middle of the list, as would be possible when calling XtGetSelectionValueIncremental repeatedly.

See Also

XtAppGetSelectionTimeout(3Xt), XtOwnSelection(3Xt)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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`.

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtNameToWidget (3Xt)

Name

XtNameToWidget, XtWidgetToWindow – translate 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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtOffset(3Xt)

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, resource lists, and other counted arrays.

See Also

XtGetResourceList(3Xt), XtSetArg(3Xt)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

Name

XtOwnSelection, XtOwnSelectionIncremental, 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*;

Boolean XtOwnSelectionIncremental(*w*, *selection*, *time*, *convert_callback*, *lose_callback*, *done_callback*, *cancel_callback*, *client_data*)
Widget *w*;
Atom *selection*;
Time *time*;
XtConvertSelectionIncrProc *convert_callback*;
XtLoseSelectionIncrProc *lose_callback*;
XtSelectionDoneIncrProc *done_callback*;
XtCancelConvertSelectionProc *cancel_callback*;
caddr_t *client_data*;

void XtDisownSelection(*w*, *selection*, *time*)
Widget *w*;
Atom *selection*;
Time *time*;

Arguments

cancel_callback
Specifies the callback procedure that is to be called to obtain the next incremental chunk of data or for each selection value obtained.

client_data Specifies the argument that is to be passed to the appropriate procedure when one of the condition occurs.

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 XtOwnSelectionIncremental informs the Intrinsics incremental selection mechanism that the specified widget believes it owns the selection. It returns True if the specified widget successfully becomes the selection owner or False otherwise.

Widgets that use the incremental transfer mechanism should use XtDisownSelection to relinquish selection ownership.

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtParseAcceleratorTable (3Xt)

Name

XtParseAcceleratorTable, XtInstallAccelerators, XtInstallAllAccelerators – manage 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 is to call `XtInstallAllAccelerators` and pass the application main window as the source.

See Also

XtParseTranslationTable(1)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtParseTranslationTable (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtPopdown (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtPopup (3Xt)

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 to pop down.
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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtRealizeWidget (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtSetArg (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtSetKeyboardFocus (3Xt)

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtSetKeyTranslator (3Xt)

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtSetSensitive (3Xt)

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtSetValues (3Xt)

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

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)
Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

XtTranslateCoordinates (3Xt)

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

Guide to the XUI Toolkit Intrinsics
Guide to the Xlib Library

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX Worksystem Software
Reference Pages,
Sections 3Dwt, 3X11, and 3Xt
AA-MA99B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

What do you like best about this manual? _____

_____

What do you like least about this manual? _____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Publications Manager
Open Software Publications Group
ZK03–2/Z04
110 SPIT BROOK ROAD
NASHUA, NH 03062–9987

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line

Reader's Comments

ULTRIX Worksystem Software
Reference Pages,
Sections 3Dwt, 3X11, and 3Xt
AA-MA99B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

What do you like best about this manual? _____

_____

What do you like least about this manual? _____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Publications Manager
Open Software Publications Group
ZK03-2/Z04
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here and Tape

Cut
Along
Dotted
Line