ULTRIX
Worksystem Software

digital

Guide to the XUI Toolkit: C Language Binding

Order Number: AA–MA95B–TE

ULTRIX Worksystem Software

Guide to the XUI Toolkit: C Language Binding

Order Number: AA-MA95B-TE

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

digital equipment corporation
maynard, massachusetts

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

Contents

About This Manual

Summary of Technical Changes

4 Menu Widget Functions

5 Dialog Box and Text Widget Functions

6 Standard Menus and Dialog Box Widget Functions

8 Cut and Paste Functions

9 Compound String Functions

10 Convenience Functions

A Widget Attributes

Figures

Tables

About This Manual

This manual describes the library of high-level and low-level C functions you use to write user interface application programs that adhere to the look and feel of the X User Interface (XUI). In addition, this manual describes other toolkit functions that you might find useful when writing your applications.

Audience

The audience for this manual includes:

- Application programmers
- Widget programmers
- Software support representatives (Digital only)
- Course instructors (Digital only)

Organization

Summary of Technical Changes

Summarizes the changes in this manual for ULTRIX Worksystem Software (UWS) Version 2.2.

Part One: Introduction

Chapter 1 Programming Considerations

Discusses the issues you need to consider when using the XUI widget functions.

Part Two: Widget Functions

Chapter 2 Window Widget Functions

Describes the components of a main window. Also discusses the functions that allow you to create instances of these widgets: main window, menu bar, window, and scroll window. In addition, describes how to add subwidget instances to the main window and scroll window widgets.

Chapter 3 Subarea Widget Functions

Discusses the functions that allow you to create instances of these widgets: scroll bar, toggle button, push button, and scale. In addition, this chapter describes convenience functions for obtaining and setting the scroll bar slider position, toggle button state, and scale slider position.

Chapter 4 Menu Widget Functions

Discusses the functions that allow you to create instances of these widgets: pull-down menu, pop-up menu, pull-down menu entry, option menu, and separator.

Chapter 5 Dialog Box and Text Widget Functions

Discusses the functions that allow you to create instances of these widgets: dialog box, pop-up dialog box, attached dialog box, pop-up attached dialog box, simple text, compound string text, color mix, and list box. In addition, this chapter describes convenience functions for manipulating the simple text widget, the compound string text widget, the color mix widget, and the list box widget.

Chapter 6 Standard Menus and Dialog Box Widget Functions

Discusses the functions that allow you to create instances of these widgets: help menu, work-in-progress box, message box, caution box, command window, selection box, and file selection box.

Chapter 7 Gadget Functions

Discusses the gadget functions, which are reduced functionality widgets.

Part Three: Other Toolkit Functions

Chapter 8 Cut and Paste Functions

Discusses functions that allow you to perform cut and paste operations.

Chapter 9 Compound String Functions

Discusses the functions you use to create and manipulate compound strings.

Chapter 10 Convenience Functions

Discusses the functions you use to display VMS messages and compound string messages. In addition, discusses a function that allows you to write upward compatible applications and widgets.

Appendix A Widget Attributes

Provides tables that list attributes supported by each widget.

Related Documentation

XUI Style Guide

Describes the standard user interface for XUI Toolkit applications, and defines the appearance, behavior, and usage of the user interface components.

XUI Programming Overview

Describes the low-level and high-level components and libraries of the X window system.

Guide to Writing Applications Using XUI Toolkit Widgets

Describes how to write applications using the XUI Toolkit.

Guide to the XUI User Interface Language Compiler

Describes the XUI User Interface Language (UIL) and its compiler.

Guide to the XUI Toolkit Intrinsics: C Language Binding

Describes the library of X intrinsics C functions used to build and manipulate widgets.

Guide to the Xlib Library: C Language Binding

Describes the library of low-level C language functions to the X Window System protocol.

Conventions

The following conventions are used in this manual:

`cat(1)`	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat(1)` indicates that you can find the material on the `cat` command in Section 1 of the reference pages.
`system output`	This typeface is used in interactive examples to indicate system output, and also in code examples and other screen displays. In text, this typeface is used to indicate the exact name of a command, option, partition, pathname, directory, or file.
filename	In syntax descriptions and function definitions, italics are used to indicate variable values; and in text, to introduce new terms or give references to other documents.

. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
mouse	The term *mouse* is used to refer to any pointing device, such as a mouse, a puck, or a stylus.
MB1, MB2, MB3	MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button. (The buttons can be redefined by the user.)

Summary of Technical Changes

This section describes the technical changes in this manual for ULTRIX Worksystem Software (UWS) Version 2.2. These technical changes also appear in the appropriate sections of the manual. The changes fall into these categories:

- Additions and changes to high-level widget and widget convenience functions
- Additions and changes to low-level widget functions
- Additions and changes to cut and paste functions
- Additions and changes to compound string functions
- New convenience functions

The manual also includes an appendix that lists the attributes for all widgets.

Additions and Changes to High-Level Widget and Widget Convenience Functions

This section describes new high-level widget and widget convenience functions, and also describes changes made to existing functions.

New High-Level Widget and Widget Convenience Functions

The following table lists the new high-level widget and high-level widget convenience functions. All these functions are described in Chapter 5.

Function	Summary Description
`DwtCSText`	Creates a compound-string text widget.
`DwtCSTextClearSelection`	Clears the global selection highlighted in the compound-string text widget.
`DwtCSTextGetEditable`	Obtains the current edit permission state indicating whether the user can edit the text in the compound-string text widget.
`DwtCSTextGetMaxLength`	Obtains the current maximum allowable length of the text in the compound-string text widget.

Function	Summary Description
DwtCSTextGetSelection	Retrieves the global selection, if any, currently highlighted, in the compound string text widget.
DwtCSTextGetString	Retrieves all text from the compound-string text widget.
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.
DwtCSTextSetEditable	Sets the permission state that determines whether the user can edit text in the compound-string text widget.
DwtCSTextSetMaxLength	Sets the maximum allowable length of the text in the compound-string text widget.
DwtCSTextSetSelection	Highlights the specified text in the compound-string text widget and makes it the current global selection.
DwtCSTextSetString	Changes the text in the compound-string text widget.
DwtColorMixGetNewColor	Returns the red, green, and blue color values to the color mixing widget.
DwtColorMixSetNewColor	Sets the new red, green, and blue color values in the color mixing widget.

Changes to High-Level Functions

Changes were made to the following high-level functions:

- DwtDialogBox
- DwtListBoxItemExists
- DwtMenu
- DwtOptionMenu
- DwtScale

Changes to DwtDialogBox

The description for the *map_callback* argument has changed to the following:

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`.

Changes to DwtListBoxItemExists – The `DwtListBoxItemExists` function no longer returns a Boolean value. The following description describes what `DwtListBoxItemExists` returns:

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.

Changes to DwtMenu – The description for the *map_callback* argument has changed as follows:

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`.

Changes to DwtOptionMenu – The `DwtOptionMenu` function now supports the *sub_menu_id* argument:

sub_menu_id Specifies the widget ID of the pull-down menu associated with the option menu during the creation phase.

Changes to DwtScale – The description for the *min_value* argument has changed as follows:

min_value Specifies the value represented by the top or left end of the scale. This argument sets the `DwtNminValue` attribute associated with `DwtScaleCreate`.

The description for the *max_value* argument has changed as follows:

max_value Specifies the value represented by the bottom or right end of the scale. This argument sets the `DwtNmaxValue` attribute associated with `DwtScaleCreate`.

Additions and Changes to Low-Level Widget Functions

This section describes new low-level widget functions, and also describes changes made to existing functions.

New Low-Level Widget Functions

The following table lists the new low-level widget functions. The `DwtColorMixCreate` and `DwtCSTextCreate` functions are described in Chapter 5. The `DwtPullEntryGadgetCreate` function is described in Chapter 7.

Function	Summary Description
`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.
`DwtCSTextCreate`	Creates a compound-string text widget.
`DwtPullEntryGadgetCreate`	Creates a pull-down menu entry gadget.

Changes to Low-Level Functions

Changes were made to the following low-level functions:

- `DwtAttachedDBCreate`
- The `DwtNdestroyCallback` core and the `DwtNdirectionRToL` common attributes
- `DwtDialogBoxCreate`
- `DwtDialogBoxPopupCreate`
- `DwtFileSelectionCreate`
- `DwtHelpCreate`
- `DwtMenuCreate`, `DwtMenuBarCreate`, `DwtOptionMenuCreate`, and `DwtRadioBoxCreate`
- `DwtMenuPopupCreate` and `DwtMenuPulldownCreate`
- `DwtMessageBoxCreate`
- `DwtPushButtonCreate`
- `DwtSTextCreate`

- DwtScaleCreate
- DwtScrollBarCreate
- DwtToggleButtonCreate

Changes to DwtAttachedDBCreate – The DwtAttachedDBCreate function now supports the DwtNresizable constraint attribute:

Attribute Name	Data Type	Default
DwtNresizable	Boolean	True

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.

Changes to the DwtNdestroyCallback and DwtNdirectionRToL Attributes – The DwtNdestroyCallback core and DwtNdirectionRToL common attributes are now described as follows:

Attribute Name	Data Type	Default
DwtNdestroyCallback	DwtCallbackPtr	NULL
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown

DwtNdestroyCallback
Specifies the callback function or functions called when the widget is about to be destroyed. Unlike all other toolkit callbacks, DwtNdestroyCallback returns only two valid arguments: *widget_id* and *tag*. The *callback_data* argument is NULL. Therefore, applications should avoid setting DwtNdestroyCallback to call general callback functions (for example, functions to handle activate, arm, disarm, and similar actions), because these functions depend on the *callback_data* argument.

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).

Changes to DwtDialogBoxCreate – The DwtNdirectionRToL widget-specific attribute is now described as follows:

Attribute Name	Data Type	Default
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown

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).

Changes to DwtDialogBoxPopupCreate – The DwtDialogBoxPopupCreate function now supports the DwtNautoUnrealize attribute. In addition, the description for the DwtNdefaultPosition attribute has changed:

Attribute Name	Data Type	Default
DwtNautoUnrealize	Boolean	False
DwtNdefaultPosition	Boolean	False

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).

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.

The description for DwtDialogBoxPopupCreate has changed to include information on the setting of colormaps. The information in the second paragraph relates to setting the colormap of a pop-up dialog box. For convenience, the entire dialog box description is included here:

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.

Changes to DwtFileSelectionCreate – The DwtFileSelectionCreate function has the following new attributes:

Attribute Name	Data Type	Default
DwtNfileToExternProc	VoidProc	NULL
DwtNfileToInternProc	VoidProc	NULL
DwtNmaskToExternProc	VoidProc	NULL
DwtNmaskToInternProc	VoidProc	NULL

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.

Changes to DwtHelpCreate – The DwtHelpCreate function no longer supports these attributes:

- DwtNhelpmessageTitle
- DwtNhelpmessageTitleType
- DwtNnulltopicMessage

The following attributes have new default values:

Attribute Name	Data Type	Default
DwtNaddtopicLabel	DwtCompString	" Additional topics"
DwtNbadframeMessage	DwtCompString	"Couldn't find frame !CS"
DwtNbadlibMessage	DwtCompString	"Couldn't open library !CS"
DwtNerroropenMessage	DwtCompString	"Error opening file !CS"
DwtNfirstTopic	DwtCompString	NULL
DwtNhelpFont	DwtFontList	Language-dependent. The American English default is "-*-TERMINAL-MEDIUM-R-NARROW--*-140-*-*-C-*-ISO8859-1"
DwtNhelpLabel	DwtCompString	"Using Help"
DwtNhistoryboxLabel	DwtCompString	"Search Topic History"
DwtNkeywordsLabel	DwtCompString	"Keyword "
DwtNnokeywordMessage	DwtCompString	"Couldn't find keyword !CS"
DwtNnotitleMessage	DwtCompString	"No title to match string !CS"
DwtNtitlesLabel	DwtCompString	"Title "
DwtNtopictitlesLabel	DwtCompString	"Topic Titles "

The following attributes for DwtHelpCreate are now described as follows:

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.

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.

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.

The DwtHelpCreate function now supports the following new attributes:

Attribute Name	Data Type	Default
DwtNcacheHelpLibrary	Boolean	False
DwtNcloseLabel	DwtCompString	"Exit"
DwtNgobacktopicLabel	DwtCompString	"Go Back"
DwtNgototopicLabel	DwtCompString	"Go To Topic"
DwtNhelpAcknowledgeLabel	DwtCompString	"Acknowledge"

Attribute Name	Data Type	Default
DwtNhelphelpLabel	DwtCompString	"Overview"
DwtNhelpOnHelpTitle	DwtCompString	"Using Help"
DwtNhelpontitleLabel	DwtCompString	"Help on "
DwtNhelptitleLabel	DwtCompString	"Help"
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNvisittopicLabel	DwtCompString	"Visit Topic"

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.

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

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

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.

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

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

Changes to DwtMenuCreate, DwtMenuBarCreate, DwtOptionMenuCreate, and DwtRadioBoxCreate – The `DwtMenuCreate` function (because it creates a work area menu) does not support the `DwtNmapCallback` and `DwtNunmapCallback` attributes; therefore, they have been deleted from the table of widget-specific attributes. In addition, these attributes have been removed from the table of inherited attributes for `DwtMenuBarCreate`, `DwtOptionMenuCreate`, and `DwtRadioBoxCreate`.

The `DwtMenuCreate` function now supports these attributes:

Attribute Name	Data Type	Default
`DwtNchangeVisAtts`	`Boolean`	`True`
`DwtNmenuExtendLastRow`	`Boolean`	`True`

`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.

Changes to DwtMenuPopupCreate and DwtMenuPulldownCreate – The `DwtMenuPopupCreate` and `DwtMenuPulldownCreate` functions now support these additional attributes:

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.

Changes to DwtMessageBoxCreate – The DwtMessageBoxCreate function now supports these attributes:

Attribute Name	Data Type	Default
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.

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.

The information in the following paragraph contains changes to the description of the DwtMessageBoxCreate function. Specifically, the changes are contained in the second paragraph, but for your convenience, the entire description is included here:

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.

Changes to DwtPushButtonCreate – The DwtPushButtonCreate function now supports this attribute:

Attribute Name	Data Type	Default
DwtNinsensitivePixmap	Pixmap	NULL

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.

Changes to DwtSTextCreate – The `DwtSTextCreate` function now supports this attribute:

Attribute Name	Data Type	Default
`DwtNuserData`	`Opaque *`	NULL

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

Changes to DwtScaleCreate – The `DwtScaleCreate` function does not support the `DwtNsliderPixmap` attribute; therefore, it has been removed from the table of widget-specific attributes.

Changes to DwtScrollBarCreate – The `DwtScrollBarCreate` function now supports this attribute:

Attribute Name	Data Type	Default
`DwtNshowArrows`	`Boolean`	`True`

`DwtNshowArrows` Specifies a boolean value that, when `True`, indicates there are arrows. If `False`, there are no arrows.

Changes to DwtToggleButtonCreate – The `DwtToggleButtonCreate` function now supports these attributes:

Attribute Name	Data Type	Default
`DwtNinsensitivePixmapOn`	`Pixmap`	NULL
`DwtNinsensitivePixmapOff`	`Pixmap`	NULL

`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.

Additions and Changes to Cut and Paste Functions

This section describes new cut and paste functions, and also describes changes made to existing functions.

New Cut and Paste Functions

Four new cut and paste functions have been added to ensure compliance with the Inter-Client Communications Conventions Manual (ICCCM). The ICCCM manual defines conventions for using the global selection mechanism that allows compliant clients to communicate with each other. The following table lists these new cut and paste functions, which are described in Chapter 8.

Function	Summary Description
`DwtStartCopyToClipboard`	Sets up storage and data structures to receive clipboard data. `DwtStartCopyToClipboard` is identical to `DwtBeginCopyToClipboard`, except that the timestamping of the event that triggered the copy is included as an argument. Because it complies with the ICCCM conventions, use of `DwtStartCopyToClipboard` is recommended over `DwtBeginCopyToClipboard`.
`DwtStartCopyFromClipboard`	Indicates that the application is ready to start copying data from the clipboard and locks the clipboard.
`DwtEndCopyFromClipboard`	Notifies the cut and paste functions that the application has completed copying an item from the clipboard and unlocks the clipboard.
`DwtClipboardRegisterFormat`	Registers the length of the data for formats not specified by ICCCM conventions.

Changes to Cut and Paste Functions

Changes were made to the following cut and paste functions:

- DwtCopyFromClipboard
- DwtInquireNextPasteCount, DwtInquireNextPasteFormat, and DwtInquireNextPasteLength

Changes to DwtCopyFromClipboard – The following status return values changed as a result of ICCCM compliance:

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.
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.

Changes to DwtInquireNextPasteCount, DwtInquireNextPasteFormat, and DwtInquireNextPasteLength – As a result of ICCCM compliance, a new status return value is possible for DwtInquireNextPasteCount, DwtInquireNextPasteFormat, and DwtInquireNextPasteLength:

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.

Additions and Changes to Compound String Functions

This section describes new compound string functions, and also describes changes made to existing functions.

New Compound String Functions

The following table lists the new compound string functions, which are described in Chapter 9.

Function	Summary Description
`DwtStringFreeContext`	Frees a compound-string context structure.
`DwtStringInitContext`	Initializes a compound-string context structure needed by `DwtGetNextSegment`

Changes to Compound String Functions

Changes were made to the following compound string functions:

- `DwtAddFontList`, `DwtCreateFontList`, and `DwtGetNextSegment`
- `DwtInitGetSegment`

Changes to DwtAddFontList, DwtCreateFontList, and DwtGetNextSegment – The `DwtAddFontList`, `DwtCreateFontList`, and `DwtGetNextSegment` functions have the following descripiton for the *charset* argument:

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`.

Changes to DwtInitGetSegment – The information in the following paragraph contains changes to the description of the `DwtInitGetSegment` function. Specifically, the changes are contained in the second and third paragraphs, but for your convenience, the entire description is included here:

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.

New Convenience Functions

The following table lists the new convenience functions, which are described in Chapter 10.

Function	Summary Description
`DwtActivateWidget`	Allows the application to simulate push button activation.
`DwtGetUserData`	Returns the user data associated with the widget.

PART ONE: INTRODUCTION

Programming Considerations 1

This chapter discusses the following issues you need to consider when using the C binding with the ULTRIX Worksystem Software XUI Toolkit widget functions:

- Methods for creating widget instances
- Widget class hierarchy
- Common attributes
- Include files
- The callback mechanism
- Function format

1.1 Methods for Creating Widget Instances

The XUI Toolkit provides three ways to create a widget instance:

- Low-level functions
- High-level functions
- XUI user interface language

Note that creating a widget instance does not cause it to be displayed on the screen. To display the widget instance on the screen after creating it, call the X intrinsics function `XtRealizeWidget`. In fact, there are a number of X intrinsics functions you can use to manipulate widget instances. For descriptions of these intrinsics functions, see the *Guide to the XUI Toolkit Intrinsics: C Language Binding*.

1.1.1 High-Level Functions

The high-level functions simplify your access to the set of available widgets and enforce the XUI "look and feel." These functions allow you to accomplish these goals by presenting you with an argument list representing the widget attributes that commonly need to be set for such objects as screen displays, menus, and scroll bars. This set of high-level functions provides much of the XUI style conforming screen display and user interface tools in building any application.

Each high-level function begins with the prefix `Dwt` followed by a name that aptly describes the widget. For example, the `DwtLabel` function creates an instance of a label widget.

1.1.2 Low-Level Functions

Unlike the high-level widget creation functions, the low-level functions provide you with access to all the widget attributes. Having access to all widget attributes makes it easier for you to customize widgets. The attributes of a particular widget include those inherited from superclass widgets and the widget's own specific attributes.

Like the high-level functions, each low-level function begins with the prefix `Dwt` followed by a name that aptly describes the widget. Unlike the high-level functions, each low-level function name ends with the word `Create`. For example, the `DwtLabelCreate` function creates an instance of a label widget.

All low-level functions have a common argument list of *parent_widget*, *name*, *override_arglist*, and *override_argcount*. For example, the function definition for `DwtLabelCreate` is:

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

Each low-level function uses a common set of attributes as well as a set of widget-specific attributes. The widget-specific attributes are described in the section for that low-level function. The common attributes are described in Section 1.3.

The declarations and meanings for the low-level function formal parameters follow:

parent_widget Specifies the parent widget ID.

name Specifies the name of the created widget.

override_arglist Specifies a list of name/value pairs that describes the attributes of the created widget. This name identifies the common or widget-specific attribute name associated with the widget. For example, the *x* argument is named `DwtNx`. The value specifies the value assigned to the previously named argument. The *override_arglist* can contain any of the attributes permitted by the widget's class hierarchy (see Section 1.2).

By specifying the inherited attributes in the *override_arglist*, you can override the default values inherited from the superclass widget. By specifying widget-specific attributes in the *override_arglist*, you can further customize your widget.

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

1.1.3 User Interface Language

The User Interface Language (UIL) compiler is a tool in the XUI Toolkit that helps you create widgets. Using the UIL, you can specify the following:

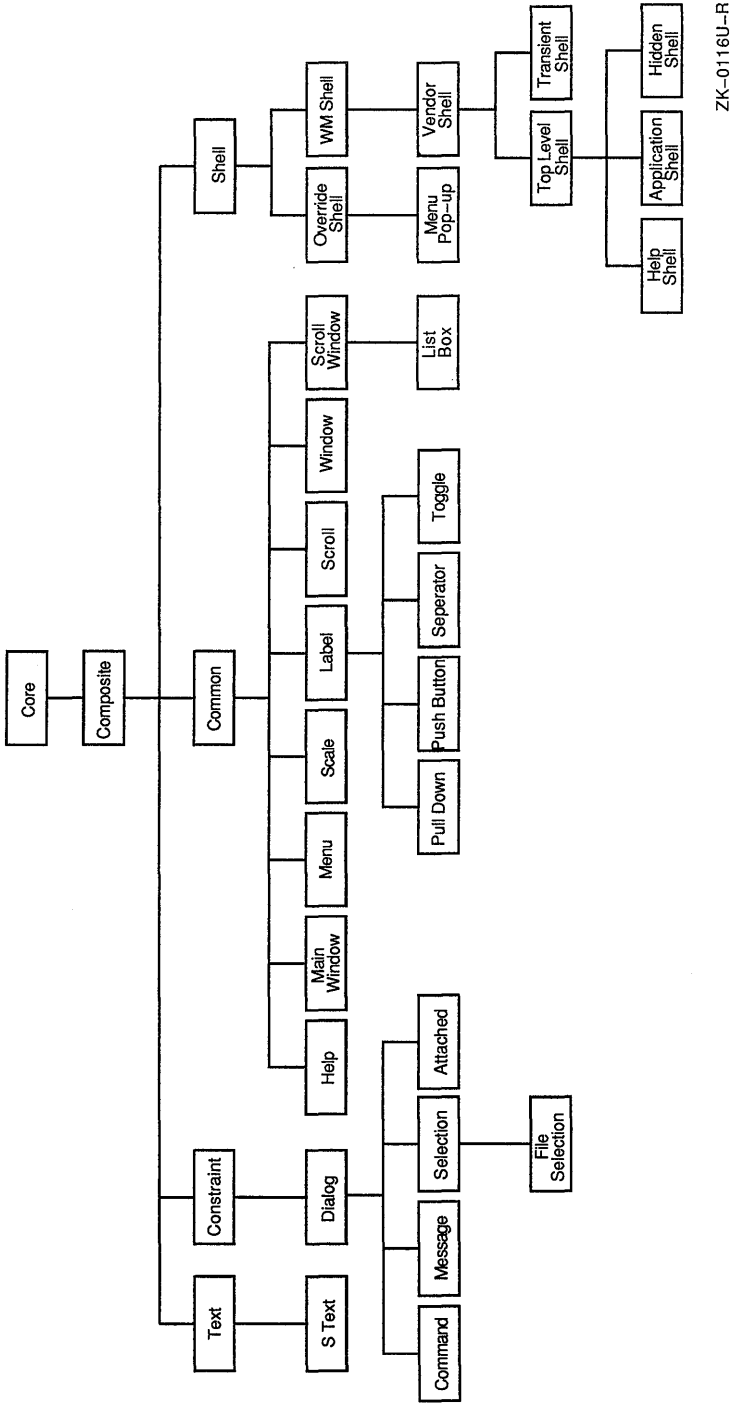
- The widgets that comprise your interface
- The characteristics (attributes) of the widgets you specify
- The hierarchy of widgets in your application.

For a description of the UIL language and compiler, see the *Guide to the XUI User Interface Language Compiler*.

1.2 Widget Class Hierarchy

Inherited attributes are determined by the widget's position in the widget class hierarchy. Within the widget class hierarchy, widgets inherit default values for attributes from all their superclass widgets. Figure 1-1 shows the widget class hierarchy.

Figure 1-1: Widget Class Hierarchy



Because attributes of several of the top-level widget classes (core, composite, common) are inherited by the majority of widgets, these attributes are called common attributes. Common attributes are described in Section 1.3.

Exceptions to the rule that widgets inherit attributes from their superclass widgets take two forms: either an inherited attribute is not supported by the widget, or the inherited attribute has a different default value than the one for its superclass widget.

As stated in the previous section, you can override the default value of inherited attributes and use widget-specific attributes to further customize your widget by specifying them in the *override_arglist*.

All widgets belong to classes that are arranged in a hierarchy. Some classes contain only one widget. For example, the push button class contains only the push button widget. Other classes contain multiple widgets. The menu and dialog classes each contain several widgets. Widget attributes reside in the widget classes.

1.3 Common Attributes

Table 1-1 lists the core widget and common widget attributes that are inherited by a majority of widgets. Following the table are descriptions of each attribute. Because the composite class has no user settable attributes, none are listed here.

Table 1-1: Core Widget and Common Widget 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

Table 1-1: (continued)

Attribute Name	Data Type	Default
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

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

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

DwtNwidth — Specifies in pixels the width of the widget window.

DwtNheight — Specifies in pixels the height of the widget window.

DwtNborderWidth — Specifies in pixels the border width of the widget window.

DwtNborder — Specifies the widget window border color.

DwtNborderPixmap — Specifies the widget window border pattern and color.

DwtNbackground — Specifies the color of background objects in the widget window.

DwtNbackgroundPixmap
: Specifies the color and pattern of background objects in the widget window.

DwtNcolormap
: Specifies the color map used for the widget's window.

DwtNsensitive
: Specifies a boolean value that, when `True`, indicates the widget reacts to input events. If `False`, the widget ignores input events.

DwtNancestorSensitive
: Specifies a boolean value that, when `True`, indicates that the parent of this widget is sensitive. If `False`, the parent of this widget is not sensitive. You can obtain the value for this attribute by calling `XtGetValues`. Your application should not explicitly set the value of this attribute.

DwtNaccelerators
: Specifies a translation table that provides an alternate mode of access to widget functions. Accelerators allow applications to define keystrokes (in addition to clicking on a screen object with the mouse) to activate a function in a widget.

DwtNdepth
: Specifies the widget window depth. This attribute is set at widget creation time, and cannot be changed by the X intrinsics function `XtSetValues`.

DwtNtranslations
: Specifies the translation table that contains the translation manager syntax for associating particular X events with particular widget events. See the *Guide to Writing Applications Using XUI Toolkit Widgets* for information on translation tables.

DwtNmappedWhenManaged
: Specifies a boolean value that, when `True`, causes the widget window to be displayed when managed. If `False`, the widget window will not be displayed when managed.

DwtNscreen
: Specifies a pointer to the Xlib `Screen` structure which contains the information about that screen and is linked to the `Display` structure. See the *Guide to the Xlib Library: C Language Binding* for information on the `Screen` structure.

DwtNdestroyCallback
: Specifies the callback function or functions called when the widget is about to be destroyed. Unlike all other toolkit callbacks, `DwtNdestroyCallback`

returns only two valid arguments: *widget_id* and *tag*. The *callback_data* argument is NULL. Therefore, applications should avoid setting DwtNdestroyCallback to call general callback functions (for example, functions to handle activate, arm, disarm, and similar actions), because these functions depend on the *callback_data* argument.

DwtNforeground — Specifies the color of foreground objects in the widget window.

DwtNhighlight — Specifies the color used for highlighting.

DwtNhighlightPixmap — Specifies the pattern and color used for highlighting.

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 the widget.

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

1.4 Include Files

To make the XUI Toolkit constants and their values available to your applications, use the include preprocessor statement to include these files:

```
#include <X11/Xlib.h>
#include <X11/DwtWidget.h>
#include <X11/DwtAppl.h>
```

1.5 The Callback Facility

The XUI widget and X intrinsics functions make it easier for you to create and display common application objects such as push buttons. However, these widget functions cannot determine your user's responses, which would

indicate a change in the widget's state. For example, you might have a file selection box that contains Ok and Cancel push buttons. When the user presses either of these push buttons you would want the application to do something appropriate. This means executing one or more functions that are private to the application. Thus, there must be some mechanism for the widget functions to ''call back'' application private functions, based on some change in the widget state. This mechanism is referred to as the callback facility. You can use this callback facility with both the high-level and the low-level functions.

At creation time (or later using `XtSetValues`), you specify in your application the callback function or functions for a widget instance. Each widget has a (possibly NULL) set of reasons for issuing callbacks, depending upon how many changes in state it is willing to communicate.

There are a number of constants that represent widget-specific callback reasons. Each begins with the prefix `Dwt` followed by some appropriate name. For example, `DwtNactivateCallback` is usually invoked when the user performs some action such as activating a push button. Examples of other constants representing widget-specific callback structures are `DwtNselectCallback`, `DwtNmapCallback`, and `DwtNhelpCallback`. Most widgets support the common `DwtNhelpCallback` attribute.

The basic structure used when specifying callback functions for a callback reason is a NULL-terminated list of the following entries:

```
typedef struct {
     void    (*proc) ();
     Opaque tag;
} DwtCallback, *DwtCallbackPtr;
```

proc Specifies a pointer to the callback procedure entry point.

tag Specifies any application supplied value. This value is usually used by the application to identify uniquely a particular widget instance, and to allow one callback function to service multiple widget instances.

By having more than one entry in a callback list for each callback reason supported by a widget, the application can specify more than one function to be called back when the appropriate widget change in state occurs.

The format for an application's callback function follows:

```
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.

tag Identifies the tag provided when the callback was specified.

callback_data Specifies a widget-specific data structure.

The format shows the declaration using `DwtAnyCallbackStruct`. However, you can also specify *callback_data* with any of the structures listed in Table 1-2.

Table 1-2: Callback Structure Names

Structure Name	Widget(s)
`DwtAnyCallbackStruct`	Main Window, Push Button, Label, Pull-Down Menu Entry, Dialog Box, Attached Dialog Box, Text, Work-in-Progress Box, Message Box, Caution Box, Help, Label Gadget, Push Button Gadget
`DwtMenuCallbackStruct`	Menu (menu work area, pull-down menu, pop-up menu, and option menu) and Menu Bar
`DwtScrollBarCallbackStruct`	Scroll Bar
`DwtTogglebuttonCallbackStruct`	Toggle Button, Toggle Button Gadget
`DwtWindowCallbackStruct`	Window
`DwtScaleCallbackStruct`	Scale
`DwtListBoxCallbackStruct`	List Box
`DwtRadioBoxCallbackStruct`	Radio Box
`DwtSelectionCallbackStruct`	Selection Box
`DwtFileSelectionCallbackStruct`	File Selection Box
`DwtCommandWindowCallbackStruct`	Command Window
`DwtCSTextCallbackStruct`	Compound String Text
`DwtColorMixCallbackStruct`	Color Mix

You can declare and pass as many of these structures as needed by the widget. Each of these structures is discussed with the appropriate functions.

Each data structure has a reason member and an event member. Note that `DwtAnyCallbackStruct` only has these two members. The other structures have additional members to supply you with the appropriate information. For example, `DwtFileSelectionCallbackStruct` has these additional members: value, value_len, dirmask, and dirmask_len. The reason member specifies the reason why this callback function was invoked. This member allows the application to invoke one callback function for a variety of reasons. The list of reasons is widget-specific; therefore, these reasons (along with the other members of the structure) are discussed with the function in the following chapters.

After you create a widget instance, your application should use `XtAddCallback`, `XtAddCallbacks`, `XtRemoveCallback`, and `XtRemoveCallbacks` to modify a widget callback list. You can use `XtSetValues` and `XtGetValues` to set the entire callback list, which may contain callbacks added by the parent widget.

1.6 Function Format

Each function and its associated formal parameters is shown according to how it is defined in the XUI Toolkit library. For example, the definition of the `DwtLabel` function and its associated formal parameters is:

```
Widget DwtLabel(parent_widget, name, x, y, label, help_callback)
      Widget parent_widget;
      char *name;
      Position x, y;
      DwtCompString label;
      DwtCallbackPtr help_callback;
```

The function definition gives you the following information:

- Return type

  If the function returns a value, it is indicated in the function definition. For `DwtLabel` the return type is a widget ID. If the function does not return a value, the data type `void` is indicated.

- Function name

  In this example, the function name is `DwtLabel`.

- Number and name of the function's formal parameters

  In this example there are six formal parameters: *parent_widget*, *name*, *x*, *y*, *label*, and *help_callback*.

- How the function's formal parameters are declared

  In this example, *parent_widget* is of type `Widget`; *name* is a pointer to a character string; *x* and *y* are of type `Position`; *label* is of type `DwtCompString`; and *help_callback* is a pointer to the structure `DwtCallbackPtr`.

  In order 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." By contrast, the explanations for all arguments that are returned to you start with the word "returns."

PART TWO: WIDGET FUNCTIONS

Window Widget Functions 2

This chapter discusses the functions you use to create instances of the following widgets:

- Application main window
- Menu bar
- Window
- Scroll window

Because you attach a menu bar to a window, it is discussed here rather than in Chapter 4. This chapter continues with a discussion of two functions: one to add subwidgets to the application main window, and the other to add a window region and a vertical or horizontal scroll bar to the scroll window.

2.1 Creating the Application Main Window

The first task of your application is to display a window. A window consists of a rectangular area with the following:

- A title bar at the top that identifies the window
- An optional menu bar directly beneath the title bar that provides access to the commands of the application
- A work area in the space remaining that displays the text and graphics related to the function of the application

The title bar is the horizontal bar that identifies your application and allows the user to manage the window. The window manager supplies the title bar for each window of your application and is the program that manages the positioning and size of windows.

The work area is the portion of the window in which users perform most application tasks. For example, if a user is working with a text editor, the work area contains the document being edited. You can create a work area widget by calling `DwtDialogBox`. For a discussion of dialog boxes and their associated functions, see Chapter 5. For more information on the appearance and operation of standard XUI windows, see the *XUI Style Guide*.

To create an instance of the main window widget, use DwtMainWindow or DwtMainWindowCreate. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtMainWindow (parent_widget, name, x, y, width, height)
      Widget parent_widget;
      char *name;
      Position x, y;
      Dimension width, height;
```

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.

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

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.

override_arglist Specifies the application override argument list.

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

The `DwtMainWindow` and `DwtMainWindowCreate` functions create an instance of the main window widget and return its associated widget ID. The main window widget can contain a menu bar region, a work area with optional scroll bars, and a command area.

The following sections discuss these aspects of the main window widget:

- Callback information
- Geometry management
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

2.1.1 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.*

2.1.2 Geometry Management

The main window widget tiles the insides of its window with up to five children as follows:

- The child widget designated as the menu bar widget is placed at the top and extends all the way across the main window. The height of the menu bar widget is set to whatever the menu requests.
- The child widget designated as the command widget is placed at the bottom and extends all the way across the main window. The height of the command widget is not altered.
- The child widget designated as the horizontal scroll bar widget is placed just above the command widget (if there is no command widget, it is placed at the bottom of the main window). The width of the horizontal scroll bar is the width of the main window minus the width of the vertical scroll bar. The height of the horizontal scroll bar is not altered.
- The child widget designated as the vertical scroll bar widget is placed on the right edge below the menu bar. The height is the distance between the bottom of the menu bar and the top of the horizontal scroll bar. The width of the vertical scroll bar is not altered.
- The child widget designated as the work area widget fills the area under the menu bar, to the left of the vertical scroll bar, and above the horizontal scroll bar. Both the width and height are altered.

There are three ways to designate that a child fill one of the previously described roles. The first is to call the `DwtMainSetAreas` function (see Section 2.5), which allows you to pass the ID of the child you want to associate with a particular role. The second is to call the X intrinsics function `XtSetValues` and set the area attributes with the appropriate widget IDs. The third method is to let the main window widget determine which child fulfills which role by using the following algorithm since only currently managed children are eligible to be designated for a role:

- A child of menu widget class (or subclass) is assumed to be the menu bar widget.
- A child of command widget class (or subclass) is assumed to be the command widget.
- A child of scroll widget class (or subclass) is either the horizontal or vertical scroll bar widget, which is determined by looking at the `DwtNorientation` attribute of the child.
- A child of any other class is assumed to be the work area.

For many applications, however, calling `DwtMainSetAreas` or `XtSetValues` is redundant, since a single main window widget may have a number of menu bars as children. Thus, by managing and unmanaging the menu bar children appropriately, the application can switch between menu bars without using `DwtMainSetAreas` or `XtSetValues`.

You can specify the size of the main window widget in two ways:

- Specify a nonzero width and height at widget creation time.

  In this case, the main window widget does not change its size on a geometry request by one of its children.

- Specify zero for both width and height at widget creation time.

  In this case, the main window widget uses the width and height of the widget designated as the work area widget in determining its width and height. The main window widget does not alter the width and height of the work area widget and places the remaining widgets based on the work area's size. The work area widget can later request a size change, and the main window widget will honor the request and reconfigure its size.

As a geometry manager, the main window widget allows the following requests to be completed:

- Changing the height of the menu bar, command window, and horizontal scroll bar widgets.
- Changing the width of the vertical scroll bar widget.
- Changing the width and height of the work area widget, if the main window widget was created with width and height of zero.

When resized, the main window widget reformats itself as described previously.

2.1.3 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the main window widget is:

- core
- composite
- common
- main window

Based on this class hierarchy, the main window widget inherits attributes from the core, composite, and common widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 2-1 lists the attributes inherited by the main window widget. For descriptions of the core and common attributes, see Chapter 1.

Table 2-1: Attributes Inherited by the Main Window Widget

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

2.1.4 Widget-Specific Attributes

Table 2-2 lists the widget-specific attributes for the main window widget. Descriptions of these attributes follow the table.

Table 2-2: Widget-Specific Attributes for the Main Window Widget

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`.

2.2 Creating the Menu Bar

The menu bar, a horizontal bar that contains lists of application commands called menus, is located immediately below the title bar and extends the full width of the window. If the space required for the menu names exceeds the window width, the menu bar will wrap to as many lines as necessary. Because menus are the principal form of command interaction for XUI applications, most applications that accept user input require a menu bar. For more detailed information about menus, see Chapter 4.

To create an instance of the menu bar widget, use `DwtMenuBar` or `DwtMenuBarCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtMenuBar(parent_widget, name, entry_callback,
                  help_callback)
    Widget parent_widget;
    char *name;
    DwtCallbackPtr entry_callback;
    DwtCallbackPtr help_callback;
```

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.

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

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*).

The DwtMenuBar and DwtMenuBarCreate functions create an instance of the menu bar widget and return its associated widget ID.

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.

The geometry management and resizing for the menu bar widget are identical to that described for the menu widget. For a discussion of geometry management and resizing, see Chapter 4. The following sections discuss these aspects of the menu bar widget:

- Callback information
- Widget class hierarchy and inherited attributes

2.2.1 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.

2.2.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the menu bar widget is:

- core
- composite
- common
- menu
- menu bar

Based on this class hierarchy, the menu bar widget inherits attributes from the core, composite, common, and menu widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Note also that the menu bar widget does not support any widget-specific attributes.

Table 2-3 lists the attributes inherited by the menu bar widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the menu attributes, see Chapter 4.

Table 2-3: Attributes Inherited by the Menu Bar Widget

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

Table 2-3: (continued)

Attribute Name	Data Type	Default
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

2.3 Creating a Window Widget

Since some XUI Toolkit applications may require a more direct interface for creating window widgets, the window widget simplifies programming by allowing you to create an application display directly in the main window widget work area. To create an instance of the window widget, use `DwtWindow` or `DwtWindowCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtWindow(parent_widget, name, x, y, width,
                 height, callback)
     Widget parent_widget;
     char *name;
     Position x, y;
     Dimension width, height;
     DwtCallbackPtr callback;
```

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`.

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

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*).

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 following sections discuss these aspects of the window widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

2.3.1 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.

2.3.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the window widget is:

- core
- composite
- common
- window

Based on this class hierarchy, the window widget inherits attributes from the core, composite, and common widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 2-4 lists the attributes inherited by the main window widget. For descriptions of the core and common attributes, see Chapter 1.

Table 2-4: Attributes Inherited by the Window Widget

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

Table 2-4: (continued)

Attribute Name	Data Type	Default
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	

2.3.3 Widget-Specific Attributes

Table 2-5 lists the widget-specific attribute for the window widget. The description of this attribute follows the table.

Table 2-5: Widget-Specific Attributes for the Window Widget

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.

2.4 Creating a Scroll Window Widget

Since some XUI Toolkit applications may require a more direct interface for creating scroll bar widgets, the scroll window widget simplifies programming by allowing you to create an application with scroll bars directly in the main window widget work area.

To create an instance of the scroll window widget, use DwtScrollWindow or DwtScrollWindowCreate. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtScrollWindow(parent_widget, name, x, y,
                       width, height)
    Widget parent_widget;
    char *name;
    Position x, y;
    Dimension width, height;
```

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.

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

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*).

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. 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.

The following sections discuss these aspects of the scroll window widget:

- Geometry management and resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

2.4.1 Geometry Management and Resizing

The scroll window widget does not do any sizing or positioning of the widget that is the work region of the scroll window widget. However, the scroll window widget does position and size the scroll bars.

The scroll window widget automatically resizes the scroll bars, but not the widget that is the work region.

2.4.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the scroll window widget is:

- core
- composite
- common
- scroll window

Based on this class hierarchy, the scroll window widget inherits attributes from the core, composite, and common widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 2-6 lists the attributes inherited by the scroll window widget. For descriptions of the core and common attributes, see Chapter 1.

Table 2-6: Attributes Inherited by the Scroll Window Widget

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

Table 2-6: (continued)

Attribute Name	Data Type	Default
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	

2.4.3 Widget-Specific Attributes

Table 2-7 lists the widget-specific attributes for the scroll window widget. Descriptions of these attributes follow the table.

Table 2-7: Widget-Specific Attributes for the Scroll Window Widget

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.

2.5 Adding Subwidgets to the Main Window

There are at least two ways to add subwidgets to the main window. The first is to create an instance of the main window widget by calling `DwtMainWindow` or `DwtMainWindowCreate` and then to use `XtSetValues` to specify the subwidget ID of the specified subwidgets (menu bar, work area, command area, and scroll bars). The second is to create an instance of the main window widget and then call the `DwtMainSetAreas` function.

```
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;
```

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`.

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.

2.6 Adding a Window Region and Scroll Bar

To add a window region and the scroll bar to the scroll bar widget window, use `DwtScrollWindowSetAreas`:

```
void DwtScrollWindowSetAreas (widget, horizontal_scroll_bar,
                              vertical_scroll_bar, work_region)
     Widget widget;
     Widget horizontal_scroll_bar;
     Widget vertical_scroll_bar;
     Widget work_region;
```

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.

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.

Subarea Widget Functions 3

Your application's work area can be divided into subareas. Subareas can contain application-related work; they can also contain controls, which are screen objects that allow the user to provide input to the application. The XUI Toolkit provides functions that allow you to create instances of subarea widgets. This chapter discusses the functions you can use to:

- Create a scroll bar widget
- Obtain and set the scroll bar slider position
- Create a label widget
- Create a toggle button widget
- Obtain and set toggle button widget state
- Create a radio box widget
- Create a push button widget
- Create a scale widget
- Obtain and set the scale slider position

3.1 Creating the Scroll Bar

The scroll bar allows the user to view information or a file that is too extensive to be displayed all at one time in a work area or another subarea. A scroll bar consists of two stepping arrows at either end of an elongated rectangle called the scroll region. A smaller rectangle called a slider resides in the scroll region. The stepping arrows and the slider allow the user to move the work area over the underlying file. The scroll region represents the underlying file. The position of the slider in the scroll region is relative to the work area's position in the underlying file.

Work areas or other subareas may have a vertical or horizontal scroll bar, or both. Horizontal scroll bars are on the bottom edge of the work area or subarea they control, and vertical scroll bars are on the right edge. For information on operating scroll bars, see the *XUI Style Guide*. To create an instance of the scroll bar widget, use `DwtScrollBar` or `DwtScrollBarCreate`. When calling `DwtScrollBar`, you set the scroll bar widget attributes presented in the formal parameter list. For `DwtScrollBarCreate`, however, you specify a list of attribute

name/value pairs that represent all the possible scroll bar widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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`.

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

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*).

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."

Note that the `DwtNtoTopCallback` and `DwtNtoBottomCallback` callbacks do not automatically set the thumb as the other callbacks do.

The following sections discuss these aspects of the scroll bar widget:

- Callback information
- Geometry management and resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

3.1.1 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.

3.1.2 Geometry Management and Resizing

The scroll bar widget does not support children.

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.

3.1.3 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the scroll bar widget is:

- core
- composite
- common
- scroll

Based on this class hierarchy, the scroll bar widget inherits attributes from the core, composite, and common widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 3-1 lists the attributes inherited by the scroll bar widget. For descriptions of the core and common attributes, see Chapter 1.

Table 3-1: Attributes Inherited by the Scroll Bar Widget

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

Table 3-1: (continued)

Attribute Name	Data Type	Default
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

3.1.4 Widget-Specific Attributes

Table 3-2 lists the widget-specific attributes for the scroll bar widget. Descriptions of these attributes follow the table.

Table 3-2: Widget-Specific Attributes for the Scroll Bar Widget

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

Table 3-2: (continued)

Attribute Name	Data Type	Default
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.

3.2 Obtaining and Setting the Scroll Bar Slider Size/Position

The ratio of the slider size to the scroll region size corresponds to the relationship between the portion of the file appearing in the window and the size of the file. For example, if 10 percent of a text file actually appears in the window, the slider takes up 10 percent of the scroll region. This provides your user with a visual cue to the size of the underlying file. Your application must maintain the ratio by keeping track of the file size and the window size. For more information on the slider size, see the *XUI Style Guide*.

The XUI Toolkit provides you with functions to obtain and set the slider size and position. To obtain the slider's currently displayed size/position values, use `DwtScrollBarGetSlider`. To change the current slider size/position value, use `DwtScrollBarSetSlider`. A description of each follows:

```
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;
```

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.

```
void DwtScrollBarSetSlider (widget, value, shown, inc,
                            page_inc, notify)
     Widget widget;
     int value;
     int shown;
     int inc, page_inc;
     Boolean notify;
```

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.

The DwtScrollBarSetSlider function sets or changes the currently displayed scroll bar widget slider for the application. 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."

3.3 Creating a Label Widget

You use labels to identify some types of controls. Labels consist of text or graphics, or both, and describe a control or set of controls. For information on using controls and labels, see the *XUI Style Guide*. To create an instance of the label widget, use `DwtLabel` or `DwtLabelCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtLabel(parent_widget, name, x, y, label, help_callback)
      Widget parent_widget;
      char *name;
      Position x, y;
      DwtCompString label;
      DwtCallbackPtr help_callback;
```

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.

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

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*).

The `DwtLabel` and `DwtLabelCreate` functions create an instance of a label widget and return its associated widget ID. 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.

The following sections discuss these aspects of the label widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

3.3.1 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.*

3.3.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the label widget is:

- core
- composite
- common
- label

Based on this class hierarchy, the label widget inherits attributes from the core, composite, and common widgets. Table 3-3 lists the attributes inherited by the label widget. For descriptions of the core and common attributes, see Chapter 1.

Table 3-3: Attributes Inherited by the Label Widget

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

Table 3-3: (continued)

Attribute Name	Data Type	Default
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

3.3.3 Widget-Specific Attributes

Table 3-4 lists the widget-specific attributes for the label widget. Descriptions of these attributes follow the table.

Table 3-4: Widget-Specific Attributes for the Label Widget

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

Table 3-4: (continued)

Attribute Name	Data Type	Default
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.

3.4 Creating a Toggle Button Widget

A toggle button is a control that can be set in either the on or off position. An individual toggle button consists of a square indicator to the left of or above the button name. When the user moves the mouse pointer onto the toggle button and clicks MB1, the button's state changes to the opposite of what it was. An empty square indicates that the toggle button is off; a filled square indicates that it is on.

To create an instance of the toggle button widget, use `DwtToggleButton` or `DwtToggleButtonCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The `DwtToggleButton` and `DwtToggleButtonCreate` functions create an instance of a toggle button widget and return its associated widget ID.

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 following sections discuss these aspects of the toggle button widget:

- Callback information

- Geometry management and resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

3.4.1 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).

3.4.2 Geometry Management and Resizing

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.

3.4.3 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the toggle button widget is:

- core
- composite
- common
- label
- toggle

Based on this class hierarchy, the toggle button widget inherits attributes from the core, composite, and common widgets. In addition, it inherits attributes from the label widget. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 3-5 lists the attributes inherited by the toggle button widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the label widget attributes, see Section 3.3.3.

Table 3-5: Attributes Inherited by the Toggle Button Widget

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

Table 3-5: (continued)

Attribute Name	Data Type	Default
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

3.4.4 Widget-Specific Attributes

Table 3-6 lists the widget-specific attributes for the toggle button widget. Descriptions of these attributes follow the table.

Table 3-6: Widget-Specific Attributes for the Toggle Button Widget

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.

3.5 Obtaining and Setting the Toggle Button Widget State

The toggle button widget has two states: on or off. There may be occasions when your application needs to know the current state of the toggle button. There may be other times when your application needs to change or set the current state of the toggle button. To obtain the state of the toggle button, use `DwtToggleButtonGetState`. To set or change the toggle button's current state, use `DwtToggleButtonSetState`. A description of each follows:

```
Boolean DwtToggleButtonGetState (widget)
     Widget widget;
```

widget Specifies the widget ID.

```
void DwtToggleButtonSetState (widget, value, notify)
     Widget widget;
     Boolean value;
     Boolean notify;
```

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.

The `DwtToggleButtonGetState` function returns the current state (value) of the toggle button, either `True` (on) or `False` (off).

The `DwtToggleButtonSetState` function sets or changes the toggle button's current state (value) within the display.

3.6 Creating a Radio Box Widget

Your application may require multiple toggle buttons. You can use the radio box widget to hold multiple toggle button widgets. To create an instance of the radio box widget, use `DwtRadioBox` or `DwtRadioBoxCreate`. 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. A description of each follows:

```
Widget DwtRadioBox(parent_widget, name, x, y,
                   entry_callback, help_callback)
    Widget parent_widget;
    char *name;
    Position x, y;
    DwtCallbackPtr entry_callback, help_callback;
```

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.

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

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*).

The `DwtRadioBox` and `DwtRadioBoxCreate` functions create an instance of the radio box widget and return its associated widget ID.

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.

The geometry management and resizing for the radio box widget are identical to that described for the menu widget. For a discussion of geometry management and resizing, see Section 4.1.1.2.

The following sections discuss these aspects of the radio box widget:

- Callback information
- Widget class hierarchy and inherited attributes

3.6.1 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.

3.6.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the radio box widget is:

- core
- composite
- common
- menu
- radio box

Based on this class hierarchy, the radio box widget inherits attributes from the core, composite, common, and menu widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 3-7 lists the attributes inherited by the radio box widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the menu widget attributes, see Section 4.1.1.4.

Table 3-7: Attributes Inherited by the Radio Box Widget

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

Table 3-7: (continued)

Attribute Name	Data Type	Default
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

Table 3-7: (continued)

Attribute Name	Data Type	Default
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

3.7 Creating a Push Button Widget

A push button consists of a name or an icon within a rectangular frame. When the user clicks MB1 anywhere on a push button, the action described by the label is initiated. For more information on push buttons, see the *XUI Style Guide*.

To create an instance of the push button widget, use `DwtPushButton` or `DwtPushButtonCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The DwtPushButton and DwtPushButtonCreate functions create an instance of the push button widget and return its associated widget ID. 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. 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. The following sections discuss these aspects of the push button widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

3.7.1 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*.

3.7.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the push button widget is:

- core
- composite
- common
- label
- push button

Based on this class hierarchy, the push button widget inherits attributes from the core, composite, and common widgets. In addition, it inherits attributes from the label widget. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 3-8 lists the attributes inherited by the push button widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the label widget attributes, see Section 3.3.3.

Table 3-8: Attributes Inherited by the Push Button Widget

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

Table 3-8: (continued)

Attribute Name	Data Type	Default
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

Table 3-8: (continued)

Attribute Name	Data Type	Default
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

3.7.3 Widget-Specific Attributes

Table 3-9 lists the widget-specific attributes for the push button widget. Descriptions of these attributes follow the table.

Table 3-9: Widget-Specific Attributes for the Push Button Widget

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.

3.8 Creating a Scale Widget

A scale allows the user to enter a value from a range of values by adjusting an arrow to a specific position along a line. Each scale consists of a label and the following elements:

- A line, with tick marks, that represents the range of the scale from *n* to *m*.
- An arrow that is the analog representation of the currently chosen scale value.
- An optional number directly opposite the arrow that is the digital representation of the currently selected scale value.

For information on designing scales, see the *XUI Style Guide*.

To create an instance of the scale widget, use `DwtScale` or `DwtScaleCreate`. When calling `DwtScale`, you set the scale widget attributes presented in the formal parameter list. For `DwtScaleCreate`, however, you specify a list of attribute name/value pairs that represent all the possible scale widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

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.

The following sections discuss these aspects of the scale widget:

- Callback structure information
- Geometry management and resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

3.8.1 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.

3.8.2 Geometry Management and Resizing

The scale widget moves its children so that they will be within the scale widget's calculated size. It is recommended that you create the scale widget with as many of the dimension-type attributes set to zero as possible. This allows the scale widget to make the best decisions for its layout.

If told to resize, the scale widget positions its children so they all fit within the scale widget.

3.8.3 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the scale widget is:

- core
- composite
- common
- scale

Based on this class hierarchy, the scale widget inherits attributes from the core, composite, and common widgets. Note that you cannot set the attributes for the composite widget; therefore, they are not shown.

Table 3-10 lists the attributes inherited by the scale widget. For descriptions of the core and common attributes, see Chapter 1.

Table 3-10: Attributes Inherited by the Scale Widget

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

Table 3-10: (continued)

Attribute Name	Data Type	Default
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

3.8.4 Widget-Specific Attributes

Table 3-11 lists the widget-specific attributes for the scale widget. Descriptions of these attributes follow the table.

Table 3-11: Widget-Specific Attributes for the Scale Widget

Attribute Name	Data Type	Default
DwtNvalue	int	zero
DwtNtitle	DwtCompString	Scale name
DwtNorientation	unsigned char	DwtOrientationHorizonta
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.

3.9 Obtaining and Setting the Scale Slider Position

The scale widget allows the user to enter a value from a range of values. There may be occasions when your application needs to know the current value of the scale. There may be other times when your application needs to change or set the current value of the scale. To obtain the current slider position value, use DwtScaleGetSlider. To set or change the current value of the slider position, use DwtScaleSetSlider. A description of each follows:

```
void DwtScaleGetSlider(widget, value_return)
     Widget widget;
     int *value_return;
```

widget Specifies the scale widget ID.

value_return Returns the current slider position value.

```
void DwtScaleSetSlider(widget, value)
     Widget widget;
     int value;
```

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`.

The `DwtScaleGetSlider` function returns the current slider position value displayed in the scale for the application.

The `DwtScaleSetSlider` function sets or changes the current slider position value within the scale widget display for the application.

Menu Widget Functions 4

A menu is a window that consists of a name and a list of choices. Menus provide the user with pointer cursor access to the tasks in your application. This chapter discusses the functions you can use to create a:

- Menu (pull-down, pop-up, and menu work area) widget
- Pull-down menu entry widget
- Option menu widget
- Separator widget

In addition, the chapter discusses some menu convenience functions you can use to position the menu when the user presses MB2 and to highlight a menu entry.

For information on menu items and for a discussion on designing and displaying the different types of menus, see the *XUI Style Guide*. For a discussion of the function used to create an instance of the menu bar widget, see Chapter 2.

4.1 Creating Menu Widgets

There are three types of menus:

- Pull-down

  A pull-down menu is the only type of menu that has a menu name. The menu-name should clearly indicate the purpose of the items in the menu. Each menu name displayed in the menu bar must be unique.

- Submenu

  A submenu is a menu that is invoked from a menu. A menu item that invokes a submenu cannot have an accelerator; menu items within a submenu, however, can have accelerators.

- Pop-up

  A pop-up menu is a menu that appears at the current pointer position. The contents of pop-up menus change with the location of the pointer cursor and the current selection. Pop-up menus contain items that can also be invoked through other mechanisms, such as pull-down menus and permanent dialog boxes. They serve as a user short-cut by reducing mouse movement.

The following are ways to create instances of the pull-down, pop-up, and menu work area widgets:

- Pull-down, pop-up, and menu work area widgets

  Call `DwtMenu` and pass the appropriate constant to indicate whether the menu is a pull-down, pop-up, or work area menu.

- Menu work area widget

  Call `DwtMenuCreate`.

- Pull-down menu widget

  Call `DwtMenuPulldownCreate`.

- Pop-up menu widget

  Call `DwtMenuPopupCreate`.

These functions are discussed in the following sections.

4.1.1 Creating Pull-Down, Pop-Up, and Menu Work Area Widgets

To create an instance of the pull-down, pop-up, and menu work area widgets, use `DwtMenu`. You can also use `DwtMenuPulldownCreate` to create an instance of the pull-down menu widget; `DwtMenuPopupCreate` to create an instance of the pop-up menu widget; and `DwtMenuCreate` to create an instance of the menu work area widget. When calling `DwtMenu`, you set the menu widget attributes presented in the formal parameter list. One of these attributes allows you to specify the type of menu widget instance you want to create: pop-up, pull-down, or menu work area. For `DwtMenuCreate`, `DwtMenuPulldownCreate`, and `DwtMenuPopupCreate`, however, you specify a list of attribute name/value pairs that represent all the possible menu widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions.

A description of each follows:

```
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;
```

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.

```
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;
```

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*).

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.

The following sections discuss these aspects of the menu widget:

- Callback information
- Geometry management and resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

4.1.1.1 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.

4.1.1.2 Geometry Management and Resizing – In general, a menu enforces positions, dimensions, and border widths for all children. In a vertical menu, entries have uniform widths — the width of the widest item in the current column. The height of each entry is not affected and is the responsibility of the item itself. In a horizontal menu, on the other hand, items have uniform height; the width is not affected.

In all menu packing modes except `DwtNmenuPackingNone`, the position of an item is completely determined by the menu; the child widget has no control of its position. In the `DwtNmenuPackingNone` packing mode, the menu does not position items.

A menu complies with all geometry requests made by its children. The menu determines the size needed to resize around its managed children and then makes the request of its geometry manager. Even if the menu's parent will not allow the request, the menu will. The child may then be clipped.

Height and width of a menu child are jointly controlled by the menu and the child. If a child requests a larger size, the menu will honor the request and then resize all other children to match. If a child requests a smaller size, the menu will honor the request; however, the menu might make the child bigger again shortly as the menu resizes the other children.

If `DwtNentryUniformBorder` is `True`, all entries will have exactly the same border width. If `DwtNentryUniformBorder` is `False`, the menu will not change any of the children's border widths.

When resized, the menu widget will lay out all its managed children in exactly the same manner as geometry management.

4.1.1.3 Widget Class Hierarchy and Inherited Attributes – The widget class hierarchy for the menu widget is:

- core
- composite
- common
- menu

The widget class hierarchy for the pull-down menu widget is:

- core
- composite
- common
- menu
- menu pulldown

The widget class hierarchy for the pop-up menu widget is:

- core
- composite
- common

- menu
- menu popup

Table 4-1 lists the attributes inherited by the menu widget. Table 4-2 lists the attributes inherited by the pull-down For descriptions of the core and common attributes, see Chapter 1. For descriptions of the menu widget attributes, see Section 4.1.1.4.

Table 4-1: 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.

Table 4-1: (continued)

Attribute Name	Data Type	Default
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

Table 4-2: 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

Table 4-2: (continued)

Attribute Name	Data Type	Default
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

Table 4-2: (continued)

Attribute Name	Data Type	Default
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

4.1.1.4 Widget-Specific Attributes – Table 4-3 lists the widget-specific attributes for the menu widget. Descriptions of these attributes follow the table.

Table 4-3: Widget-Specific Attributes for the Menu Widget

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)

Table 4-3: (continued)

Attribute Name	Data Type	Default
		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.

Table 4-4 lists the widget-specific attributes for the pull-down and pop-up menu widgets. Descriptions of these attributes follow the table.

Table 4-4: Pull-Down Menu and Pop-Up Menu Widgets

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.

4.2 Creating Pull-Down Menu Entry Widgets

To create an instance of the pull-down menu entry widget, use `DwtPullDownMenuEntry` or `DwtPullDownMenuEntryCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

`Widget DwtPullDownMenuEntryCreate` (*parent_widget, name, override_arglist, override_argcount*)

```
Widget parent_widget;
char *name;
ArgList override_arglist;
int override_argcount;
```

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*).

The `DwtPullDownMenuEntry` and `DwtPullDownMenuEntryCreate` functions create an instance of the pull-down menu entry widget and return its associated widget ID. 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.

The following sections discuss these aspects of the pull-down menu entry widget:

- Callback information
- Geometry management and resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

4.2.1 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*.

4.2.2 Geometry Management and Resizing

The pull-down menu entry widget may have a captive push button child, the hotspot widget. This is the only child it allows. It refuses all other requests.

When the pull-down menu entry widget is resized by its parents, it calls `DwtLabelCreate`'s resize procedure and then repositions and resizes the hotspot widget.

4.2.3 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the pull-down menu entry widget is:

- core
- composite
- common
- label
- pulldown menu entry

Table 4-5 lists the attributes inherited by the pull-down menu entry widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the label widget attributes, see Section 3.3.3.

Table 4-5: Attributes Inherited by the Pull-Down Menu Entry Widget

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

Table 4-5: (continued)

Attribute Name	Data Type	Default
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

4.2.4 Widget-Specific Attributes

Table 4-6 lists the widget-specific attributes for the pull-down menu entry widget. Descriptions of these attributes follow the table.

Table 4-6: Widget-Specific Attributes for the Pull-Down Menu Entry Widget

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.

4.3 Creating an Option Menu Widget

To create an instance of the option menu widget, use `DwtOptionMenu` or `DwtOptionMenuCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The DwtOptionMenu and DwtOptionMenuCreate functions create an instance of the option menu widget and return its associated widget ID. 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.

The following sections discuss these aspects of the option menu widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

The option menu widget follows the same rules for geometry management and resizing as its superclass the menu widget. For information on geometry management and resizing, see Section 4.1.1.2.

4.3.1 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.

4.3.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the option menu widget is:

- core
- composite
- common
- menu
- option menu

Table 4-7 lists the attributes inherited by the option menu widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the menu widget attributes, see Section 4.1.1.4.

Table 4-7: Attributes Inherited by the Option 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`	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`

Table 4-7: (continued)

Attribute Name	Data Type	Default
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

Table 4-7: (continued)

Attribute Name	Data Type	Default
		`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`

4.3.3 Widget-Specific Attributes

Table 4-8 lists the widget-specific attributes for the option menu widget. Descriptions of these attributes follow the table.

Table 4-8: Widget-Specific Attributes for the Option Menu Widget

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.

4.4 Menu Convenience Functions

To position the menu when the user presses MB2, use `DwtMenuPosition`.

```
void DwtMenuPosition(position, event)
     Widget position;
     XEvent *event;
```

position Specifies the position of the menu.

event Specifies the event passed to the action procedure which manages the pop-up menu.

The `DwtMenuPosition` function positions the menu when the user presses MB2. This must be called before managing the pop-up menu.

To keep an entry highlight after the user clicks on a menu item, use `DwtPullDownMenuEntryHilite`.

```
void DwtPullDownMenuEntryHilite(pulldown, highlight)
     Widget pulldown;
     int highlight;
```

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.

The `DwtPullDownMenuEntryHilite` function keeps an entry highlight after the user clicks on a menu item.

4.5 Creating a Separator Widget

To create an instance of the separator widget, use `DwtSeparator` or `DwtSeparatorCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtSeparator(parent_widget, name, x, y, orientation)
     Widget parent_widget;
     char *name;
     Position x, y;
     unsigned char orientation;
```

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.

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

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*).

The DwtSeparator and DwtSeparatorCreate functions create an instance of the separator widget and return its associated widget ID. 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.

The following sections discuss these aspects of the separator widget:

- Widget class hierarchy and inherited attributes
- Widget-specific attributes

4.5.1 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the separator widget is:

- core
- composite
- common
- label
- separator

Table 4-9 lists the attributes inherited by the separator widget. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the label widget attributes, see Section 3.3.3.

Table 4-9: Attributes Inherited by the Separator Widget

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

Table 4-9: (continued)

Attribute Name	Data Type	Default
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

4.5.2 Widget-Specific attributes

Table 4-10 lists the widget-specific attribute for the separator widget. The description of this attribute follows the table.

Table 4-10: Widget-Specific Attribute for the Separator Widget

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.

Dialog Box and Text Widget Functions 5

A dialog box is a window that solicits input from the user and displays messages. The user responds to the input requests or messages by means of controls within the dialog box. Your application creates dialog boxes, and the window manager manages them. This chapter discusses functions you can use to:

- Create a dialog box widget
- Create an attached dialog box widget
- Create a simple text widget
- Manipulate a simple text widget
- Create a compound string text widget
- Manipulate a compound string text widget
- Create a color mix widget
- Manipulate a color mix widget
- Create a list box widget

The chapter concludes with a section that describes list box convenience functions.

5.1 Creating a Dialog Box Widget

Dialog boxes are displayed in response to some user command to obtain clarification or to notify the user. A dialog box may solicit the user for additional information so it can execute a command. A dialog box may also notify the user that a command was incorrectly entered, caution the user about a potential problem, or notify the user of some unusual event.

To allow the user to enter input and to perform other application-related tasks, your dialog boxes will probably make use of the controls found in control panels. In addition, your dialog boxes may make use of other controls:

- Push buttons
- Text entry fields

- Option menus
- List boxes

Dialog boxes are either modal or modeless. A modal dialog box stops the work session and solicits input from the user. A modeless dialog box also solicits input from the user but does not stop the user's work or the activity of any application, including the application that displayed the dialog box. For more information on modal and modeless dialog boxes, and for discussions on displaying dialog boxes, working with dialog box controls and labels, and designing dialog boxes, see the *XUI Style Guide*.

The following are ways to create instances of the dialog box widget:

- For modal, modeless, and work area dialog box widgets call `DwtDialogBox` and pass the appropriate constant to the *style* argument.
- For modal and modeless dialog box widgets call `DwtDialogBoxPopupCreate` and pass the appropriate constant to `DwtNstyle`.
- For work area dialog box widgets call `DwtDialogBoxCreate` and pass the appropriate constant to `DwtNstyle`.

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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

```
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;
```

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*).

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.

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.

The following sections discuss these aspects of the dialog box and pop-up dialog box widgets:

- Callback information
- Geometry management
- Resizing
- Widget class hierarchy and inherited attributes
- Widget-specific attributes
- Constraint attributes

5.1.1 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*.

5.1.2 Geometry Management

The dialog box (or pop-up dialog box) widget is a generic container that treats all its children equally in terms of geometry management. When a child is first added to a dialog box, it is placed using the DwtNx, DwtNy, DwtNwidth, DwtNheight, and DwtNborderWidth core attributes specified in the widget. The dialog box does not override any of the geometry of its children.

The value of the DwtNchildOverlap attribute affects how the geometry manager reacts to geometry requests from its children. If True (the default), the dialog box approves a request from a child, even if the request results in the child overlapping another child of the dialog box. If False, the dialog box geometry manager denies such requests.

5.1.3 Resizing

The resizing behavior of the dialog box (or pop-up dialog box) widget is controlled by the DwtNresize attribute. See the description of that attribute.

5.1.4 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the dialog box and pop-up dialog box widgets is:

- core

- composite
- constraint
- dialog

Based on this class hierarchy, the dialog box and pop-up dialog box widgets inherit attributes from the core, composite, and constraint widgets. Note that you cannot set the attributes for the constraint widget; therefore, they are not shown. The inherited attributes are listed in the following tables.

Table 5-1 lists the attributes inherited by the dialog box and pop-up dialog box widget. For descriptions of the core attributes, see Chapter 1.

Table 5-1: Attributes Inherited by the Dialog Box and Pop-Up Dialog Box Widgets

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

5.1.5 Widget-Specific Attributes

Table 5-2 lists the widget-specific attributes for the dialog box and pop-up dialog box widgets. Table 5-3 lists additional widget-specific attributes supported only by the pop-up dialog box widget. Descriptions of these attributes follow the tables.

Table 5-2: Widget-Specific Attributes for the Dialog Box and Pop-Up Dialog Box Widgets

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 Tal key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT()\n' Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()";

Table 5-3: 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.

DwtNdirectionRToLSpecifies 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`).

5.1.6 Constraint Attributes

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`.

5.2 Creating an Attached Dialog Box Widget

The following are ways to create instances of the attached dialog box widget:

- For modal, modeless, and work area attached dialog box widgets call `DwtAttachedDB` and pass the appropriate constant to the *style* argument.
- For modal and modeless attached dialog box widgets call `DwtAttachedDBPopupCreate`.
- For work area attached dialog box widget call `DwtAttachedDBCreate`.

When calling `DwtAttachedDB`, you set the attached dialog box widget attributes presented in its formal parameter list. For `DwtAttachedDBCreate` and `DwtAttachedDBPopupCreate`, however, you specify a list of attribute name/value pairs that represent all the possible attached dialog box widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

```
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;
```

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*).

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.

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.

The following sections discuss these aspects of the attached dialog box widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes
- Constraint attributes

The attached dialog box widget and attached dialog box pop-up widget follow the same rules for geometry management and resizing as their superclass, the dialog box widget. For information on dialog box widget geometry management and resizing, see Section 5.1.2.

5.2.1 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*.

5.2.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the attached dialog box widget is:

- core
- composite
- constraint

- dialog
- attached dialog

The widget class hierarchy for the attached dialog box pop-up widget is:

- core
- composite
- constraint
- dialog
- attached dialog box popup

Table 5-4 lists the attributes inherited by the attached dialog box and attached dialog box pop-up widgets. For descriptions of the core and common attributes, see Chapter 1. For descriptions of the dialog widget attributes, see Section 5.1.5.

Table 5-4: Attributes Inherited by the Attached Dialog Box and Attached Dialog Box Pop-Up Widgets

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
DwtNancestorSensitive	Boolean	The bitwise AND of the par widget's DwtNsensitive DwtNancestorSensiti attributes
DwtNaccelerators	XtTranslations	NULL
DwtNdepth	int	Depth of the parent window
DwtNtranslations	XtTranslations	NULL
DwtNmappedWhenManaged	Boolean	True

Table 5-4: (continued)

Attribute Name	Data Type	Default
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.

Table 5-4: (continued)

Attribute Name	Data Type	Default
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

Table 5-4: (continued)

Attribute Name	Data Type	Default
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()";

5.2.3 Widget-Specific Attributes

Table 5-5 lists the widget-specific attributes for the attached dialog box and attached dialog box pop-up widgets. Descriptions of these attributes follow the table.

Table 5-5: Widget-Specific Attributes for the Attached Dialog Box and Attached Dialog Box Pop-Up Widgets

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.

5.2.4 Constraint Attributes

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.

5.3 Creating a Simple Text Widget

To create an instance of the simple text widget, use `DwtSText` or `DwtSTextCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

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

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. 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`.

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

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*).

The `DwtSText` and `DwtSTextCreate` functions create an instance of a simple text widget and return its associated widget ID. 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.

The following sections discuss these aspects of the simple text widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

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

5.3.1 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*.

5.3.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the simple text widget is:

- core
- text
- stext

Table 5-6 lists the attributes inherited by the simple text widget. For descriptions of the core attributes, see Chapter 1. For descriptions of the core and common attributes, see Chapter 1.

Table 5-6: Attributes Inherited by the Simple Text Widget

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

Table 5-6: (continued)

Attribute Name	Data Type	Default
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

5.3.3 Widget-Specific Attributes

Table 5-7 lists the widget-specific attributes for the simple text widget. Descriptions of these attributes follow the table.

Table 5-7: Widget-Specific Attributes for the Simple Text Widget

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

Table 5-7: (continued)

Attribute Name	Data Type	Default
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.

5.4 Manipulating a Simple Text Widget

The XUI Toolkit provides you with some useful functions with which you can manipulate the text widget. Specifically, these functions allow you to:

- Clear the global selection
- Obtain the global selected text
- Set the global selected text
- Obtain a text string
- Display a new text string
- Obtain the current maximum length of the text widget

- Set the maximum length of the text widget
- Obtain editing information about the text
- Set the editing permission for the text
- Replace part of the old text

5.4.1 Clearing, Obtaining, and Setting the Global Selection

To clear the global selection, use `DwtSTextClearSelection`.

```
void DwtSTextClearSelection (widget, time)
     Widget widget;
     Time time;
```

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.

The `DwtSTextClearSelection` function clears the global selection highlighted in the simple text widget.

To retrieve the global selection currently highlighted, use `DwtSTextGetSelection`:

```
char *DwtSTextGetSelection (widget)
Widget widget;
```

widget Specifies the widget ID.

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`.

To set the selected global text into the widget, use `DwtSTextGetSelection`:

```
void DwtSTextSetSelection (widget, first, last, time)
     Widget widget;
     int first, last;
     Time time;
```

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.

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.

5.4.2 Obtaining and Displaying a New Text String

To obtain the pointer to the currently displayed string, use `DwtSTextGetString`:

```
char  *DwtSTextGetString(widget)
      Widget widget;
```

widget Specifies the ID of the simple text widget

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`.

To set the text string in the simple text widget, use `DwtSTextSetString`:

```
void DwtSTextSetString(widget, value)
     Widget widget;
     char *value;
```

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.

The `DwtSTextSetString` function completely changes the string in the simple text widget.

5.4.3 Obtaining and Setting the Maximum Length of the Simple Text Widget

To obtain the current maximum allowable length of the text string in the simple text widget, use `DwtSTextGetMaxLength`:

```
int DwtSTextGetMaxLength(widget)
    Widget widget;
```

widget Specifies the ID of the simple text widget whose maximum text string length you want to obtain.

The `DwtSTextGetMaxLength` function returns the current maximum allowable length of the text string in the simple text widget.

To set the maximum length of the text string in the simple text widget, use `DwtSTextSetMaxLength`:

```
void DwtSTextSetMaxLength(widget, max_length)
    Widget widget;
    int max_length;
```

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`.

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.

5.4.4 Obtaining and Setting Editing Information About the Text

To obtain the edit-permission-state in the simple text widget, use `DwtSTextGetEditable`:

```
Boolean DwtSTextGetEditable(widget)
    Widget widget;
```

widget Specifies the ID of the simple text widget whose edit permission state you want to obtain.

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.

To set the edit permission state in the simple text widget, use `DwtSTextSetEditable`:

```
void DwtSTextSetEditable(widget, editable)
     Widget widget;
     Boolean editable;
```

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.

The `DwtSTextSetEditable` function sets the edit permission state information concerning whether the user can edit text in the simple text widget.

5.4.5 Replacing Part of the Old Text

To replace part of the existing text string, use `DwtSTextReplace`:

```
void DwtSTextReplace(widget, from_pos, to_pos, value)
     Widget widget;
     int from_pos, to_pos;
     DwtCompString value;
```

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.

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.

5.5 Creating a Compound String Text Widget

To create an instance of the compound string text widget, use `DwtCSText` or `DwtCSTextCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
Widget DwtCSText (parent_widget, name, x, y, cols, rows, value)
```

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`.

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

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*).

The `DwtCSText` and `DwtCSTextCreate` functions create an instance of a compound-string text widget and return its associated widget ID. 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.

5.5.1 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.

5.5.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the simple text widget is:

- core
- text
- cstext

Table 5-8 lists the attributes inherited by the compound string text widget. For descriptions of the core attributes, see Chapter 1.

Table 5-8: Attributes Inherited by the Compound String Text Widget

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

Table 5-8: (continued)

Attribute Name	Data Type	Default
DwtNscreen	Screen *	The parent screen
DwtNdestroyCallback	DwtCallbackPtr	NULL

5.5.3 Widget-Specific Attributes

Table 5-9 lists the widget-specific attributes for the compound string text widget. Descriptions of these attributes follow the table.

Table 5-9: Widget-Specific Attributes for the Compound String Text Widget

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

Table 5-9: (continued)

Attribute Name	Data Type	Default
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.

5.6 Manipulating a Compound String Text Widget

The XUI Toolkit provides you with some useful functions with which you can manipulate the compound string text widget. Specifically, these functions allow you to:

- Clear the global selection
- Obtain and set the global selected compound string text
- Obtain and display a compound string text
- Obtain and set the current maximum length of the compound string text widget
- Obtain and set editing information about the compound string text widget
- Replace part of the old compound string text widget

5.6.1 Clearing, Obtaining, and Setting the Global Selection

To clear the global selection in the compound string text widget, use DwtCSTextClearSelection.

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

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.

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

To retrieve the global selection in the compound string text widget, use DwtCSTextGetSelection.

```
DwtCompString DwtCSTextGetSelection(widget)
Widget widget;
```

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

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.

To set the specified text in the compound string text widget, use DwtCSTextSetSelection.

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

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.

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.

5.6.2 Obtaining and Displaying a New Compound String Text

To retrieve all the text from the compound string text widget, use DwtCSTextGetString.

```
DwtCompString DwtCSTextGetString(widget)
     Widget widget;
```

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

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.

To set the text in the compound string text widget, use DwtCSTextSetString.

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

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.

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

5.6.3 Obtaining and Setting the Maximum Length of the Compound String Text Widget

To obtain the maximum length of the text in the compound string text widget, use DwtCSTextGetMaxLength.

```
int DwtCSTextGetMaxLength(widget)
     Widget widget;
```

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

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

To set the maximum length of the text in the compound string text widget, use DwtCSTextSetMaxLength.

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

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.

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.

5.6.4 Obtaining and Setting Editing Information About the Compound String Text Widget

To obtain the edit permission state in the compound string text widget, use DwtCSTextGetEditable.

```
Boolean DwtCSTextGetEditable(widget)
```

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

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.

To set the edit permission state in the compound string text widget, use DwtCSTextSetEditable.

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

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.

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

5.6.5 Replacing Part of the Old Text in the Compound String Text Widget

To replace part of the existing text in the compound string text widget, use DwtCSTextReplace.

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

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.

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.

5.7 Creating a Color Mix Widget

To create an instance of the color mix widget, use `DwtColorMixCreate`. When calling `DwtColorMixCreate`, you specify a list of attribute name/value pairs that represents all the possible color mixing widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

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

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*).

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.

5.7.1 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.

5.7.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the color mix widget is:

- core
- composite
- constraint
- dialog

Table 5-10 lists the attributes inherited by the color mix widget. For descriptions of the core and common attributes, see Chapter 1.

Table 5-10: Attributes Inherited by the Color Mix Widget

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

Table 5-10: (continued)

Attribute Name	Data Type	Default
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()";

5.7.3 Widget-Specific Attributes

Table 5-11 lists the widget-specific attributes for the color mix widget. Descriptions of these attributes follow the table.

Table 5-11: Widget-Specific Attributes for the Color Mix Widget

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"

Table 5-11: (continued)

Attribute Name	Data Type	Default
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.

Table 5-11: (continued)

Attribute Name	Data Type	Default
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.

5.8 Manipulating a Color Mix Widget

The XUI Toolkit provides you with some useful functions with which you can manipulate the color mix widget. Specifically, these functions allow you to obtain and set the red, green, and blue color values.

To obtain the red, green, and blue color values for the color mix widget, use DwtColorMixGetNewColor.

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

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.

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.

To set the red, green, and blue color values in the color mix widget, use DwtColorMixSetNewColor.

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

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.

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.

5.9 Creating a List Box Widget

A list box displays a number of choices, such as names of available files, from which the user can choose. The list box consists of items within a rectangular area and a scroll bar to the right of those items. The user views the items by operating the scroll bar. To create an instance of the list box widget, use DwtListBox or DwtListBoxCreate.

When calling DwtListBox, you set the list box widget attributes presented in the parameter list. For DwtListBoxCreate, however, you specify a list of attribute name/value pairs that represent all the possible list box widget attributes.

After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

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.

The following sections discuss these aspects of the list box widget:

- Callback information
- Geometry management and resizing

- Widget class hierarchy and inherited attributes
- Widget-specific attributes

5.9.1 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.

5.9.2 Geometry Management and Resizing

The list box widget does not support children; therefore, there is no geometry semantics.

The size of the list box is determined by the following attributes in descending precedence:

- `DwtNheight` and `DwtNwidth`
- `DwtNvisibleItemsCount`
- `DwtNresize`

Setting the core attributes `DwtNheight` and `DwtNwidth` overrides the widget-specific default settings. The following describes the sizing option.

The default list box height is determined by `DwtNvisibleItemsCount`. Once set, the list box height will not change, regardless of the number of items the list box actually contains, unless the core attribute `DwtNheight` or the widget-specific attribute `DwtNvisibleItemsCount` is modified. It is recommended that you control list box height by setting `DwtNvisibleItemsCount` rather than `DwtNheight`.

The default list box width is controlled by the `DwtNresize` attribute. By default, `DwtNresize` is `True`, and the list box increases its width to accommodate items wider than its current width. However, the list box does not shrink if wider items are removed. Note that only increases, not decreases, in width are requested. The list box will not shrink if wider items are removed. You can control this behavior with the `DwtNresize` attribute. If `DwtNresize` is `False`, the width remains constant.

To keep the width of the list box constant, set `DwtNwidth` to the desired width and set `DwtNresize` to `False`. In addition, set `DwtNhorizontal` to `True` so that users can scroll the item list horizontally to see items wider than the list box.

5.9.3 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the list box widget is:

- core
- composite
- common
- scroll window
- list box

Based on this class hierarchy, the list box widget inherits attributes from the core, composite, and common widgets. In addition, it inherits attributes from the scroll window widget.

Table 5-12 lists the attributes inherited by the list box widget. For descriptions of the core and common attributes, see Chapter 1.

Table 5-12: Attributes Inherited by the List Box Widget

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

Table 5-12: (continued)

Attribute Name	Data Type	Default
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	DwtDirectionRightDow
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	NOT SUPPORTED	
Scroll Window Attributes		
DwtNhorizontalScrollBar	Widget	NULL
DwtNverticalScrollBar	Widget	NULL
DwtNworkWindow	Widget	NULL
DwtNshownValueAutomaticHoriz	Boolean	True
DwtNshownValueAutomaticVert	Boolean	False

5.9.4 Widget-Specific Attributes

Table 5-13 lists the widget-specific attributes for the list box widget. Descriptions of these attributes follow the table.

Table 5-13: Widget-Specific Attributes for the List Box Widget

Attribute Name	Data Type	Default
DwtNmarginWidth	Dimension	10 pixels
DwtNmarginHeight	Dimension	4 pixels
DwtNspacing	Dimension	1 pixel
DwtNitems	DwtCompString *	NULL
DwtNitemsCount	int	Zero

Table 5-13: (continued)

Attribute Name	Data Type	Default
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.

5.10 List Box Convenience Functions

This section discusses the functions you can use to:

- Add and delete items to a list box widget
- Delete an item by position
- Deselect a single or all previously selected items
- Verify the existence of an item
- Select an item in the list box
- Set the horizontal position
- Make an item the first visible item in the list box
- Make a position the top visible position in the list box
- Select and deselect an item identified by its position in the list box

5.10.1 Adding and Deleting Items to a List Box Widget

The XUI Toolkit provides functions with which you can add or delete items in a list box. To add an item to a list box, use DwtListBoxAddItem.

```
void DwtListBoxAddItem(widget, item, position)
     Widget widget;
     DwtCompString item;
     int position;
```

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.

The `DwtListBoxAddItem` function adds an item to a list within the list box widget.

To delete an item from a list box, use `DwtListBoxDeleteItem`:

```
void DwtListBoxDeleteItem(widget, item)
     Widget widget;
     DwtCompString item;
```

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.

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.

5.10.2 Deleting an Item By Position

To delete an item identified by its position from a list box, use `DwtListBoxDeletePos`:

```
void DwtListBoxDeletePos(widget, position)
     Widget widget;
     int position;
```

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.

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.

5.10.3 Deselecting a Single Item or All Previously Selected Items

To deselect a previously selected item in a list box, use DwtListBoxDeselectItem:

```
void DwtListBoxDeselectItem(widget, item)
     Widget widget;
     DwtCompString item;
```

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).

The DwtListBoxDeselectItem function deselects (removes highlighting) an item previously selected, and removes it from the list of selected items.

To deselect all the previously selected items in a list box, use DwtListBoxDeselectAllItems:

```
void DwtListBoxDeselectAllItems(widget)
     Widget widget;
```

widget Specifies the ID of the list box widget from whose list you want to delete all previously selected items.

The DwtListBoxDeselectAllItems function deselects (removes highlighting) all items previously selected, and removes them from the list of selected items.

5.10.4 Verifying the Existence of an Item

To verify the existence of a particular item in a list box, use DwtListBoxItemExists:

```
int DwtListBoxItemExists(widget, item)
     Widget widget;
     DwtCompString item;
```

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.

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.

5.10.5 Selecting an Item in the List Box

To select an item in the list box, use DwtListBoxSelectItem.

```
void DwtListBoxSelectItem(widget, item, notify)
     Widget widget;
     DwtCompString item;
     Boolean notify;
```

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.

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.

5.10.6 Setting the Horizontal Position

To set the horizontal position to a specified position, use DwtListBoxSetHorizPos:

```
void DwtListBoxSetHorizPos(widget, position)
     Widget widget;
     int position;
```

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.

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.

5.10.7 Making an Item the First Visible Item in the List Box

To make a specified item the first visible item in a list box, use DwtListBoxSetItem:

```
void DwtListBoxSetItem(widget, item)
     Widget widget;
     DwtCompString item;
```

widget Specifies the widget ID.

item Specifies the item to be made the first item in the list box.

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.

5.10.8 Making a Position the Top Visible Position in the List Box

To make a specified position the top visible position in a list box, use DwtListBoxSetPos:

```
void DwtListBoxSetPos(widget, position)
     Widget widget;
     int position;
```

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.

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.

5.10.9 Selecting and Deselecting an Item by Its Positiion in the List Box

To select an item identified by its position in the list box, use DwtListBoxSelectPos.

```
void DwtListBoxSelectPos (widget, position, notify)
     Widget widget;
     int position;
     Boolean notify;
```

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.

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.

To deselect an item identified by its position in the list box, use DwtListBoxDeselectPos.

```
void DwtListBoxDeselectPos (widget, position)
     Widget widget;
     int position;
```

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.

The DwtListBoxDeselectPos function deselects an item (removes highlighting) based on its position in a list box and removes the item from the selected list.

Standard Menus and Dialog Box Widget Functions 6

Many applications perform common tasks such as creating, modifying, or editing files. To manage these common tasks, you will probably implement standard menus and dialog boxes. The XUI Toolkit provides functions that allow you to create instances of standard menus and standard dialog boxes. This chapter discusses the functions you can use to:

- Create a help menu widget
- Create a work-in-progress box widget
- Create a message box widget
- Create a caution box widget
- Create a command window widget
- Manipulate the command line
- Create a selection box widget
- Create a file selection box widget
- Initiate a search with a directory mask option

For guidelines on what to follow when implementing standard menus and standard dialog boxes, see the *XUI Style Guide*.

6.1 Creating the Help Menu Widget

XUI application programmers are strongly encouraged to provide user assistance as part of your applications. Applications that provide online assistance to users should have a pull-down Help menu. For information on invoking and designing the Help menu, see the *XUI Style Guide*.

To create an instance of the help menu widget, use `DwtHelp` or `DwtHelpCreate`. When calling `DwtHelp`, you set the help menu widget attributes presented in the formal parameter list. For `DwtHelpCreate`, however, you specify a list of attribute name/value pairs that represent all the possible help menu widget attributes. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

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.

The following sections discuss these aspects of the help menu widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.1.1 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*.

6.1.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the help menu widget is:

- core
- composite
- common
- help

Based on this class hierarchy, the help menu widget inherits attributes from the core, composite, and common widgets.

Table 6-1 lists the attributes inherited by the help menu widget. For descriptions of the core and common attributes, see Chapter 1.

Table 6-1: Attributes Inherited by the Help Widget

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

Table 6-1: (continued)

Attribute Name	Data Type	Default
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

6.1.3 Widget-Specific Attributes

Table 6-2 lists the attributes inherited by the help menu widget.

Table 6-2: Widget-Specific Attributes for the Help Widget

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"

Table 6-2: (continued)

Attribute Name	Data Type	Default
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`.

6.2 Creating the Work-in-Progress Box Widget

Your application can indicate that a time-consuming operation is taking place by using a standard dialog box called the work-in-progress box. To create an instance of the work-in-progress box widget, use DwtWorkBox or DwtWorkBoxCreate. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The `DwtWorkBox` and `DwtWorkBoxCreate` functions create an instance of a work-in-progress box widget and return its associated widget ID. 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`.

The work-in-progress box widget follows the same rules for geometry management and resizing as its superclass the dialog pop-up widget. For

information on geometry management, see Section 5.1.2; for information on resizing, see Section 5.1.3.

The following sections discuss these aspects of the work-in-progress box widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.2.1 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*.

6.2.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the work-in-progress box is:

- core
- composite
- constraint
- dialog
- message

Based on this class hierarchy, the work-in-progress box widget inherits attributes from the core, composite, constraint, and dialog box widgets. Note that the work-in-progress box is a message class widget. Note also that you cannot set the attributes for the composite or constraint widgets; therefore, they are not shown.

Table 6-3 lists the attributes inherited by the work-in-progress box widget. For descriptions of the core attributes, see Chapter 1. For descriptions of the dialog widget attributes, see Section 5.1.5.

Table 6-3: Attributes Inherited by the Work-in-Progress Box Widget

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

Table 6-3: (continued)

Attribute Name	Data Type	Default
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	

6.2.3 Widget Class Hierarchy and Inherited Attributes

Table 6-4 lists the widget-specific attributes for the work-in-progress box widget. Descriptions of these attributes follow the table.

Table 6-4: Widget-Specific Attributes for the Work-in-Progress Box Widget

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNcancelLabel	DwtCompString	"Cancel"
DwtNcancelCallback	DwtCallbackPtr	NULL

Table 6-4: (continued)

`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`.

6.3 Creating a Message Box Widget

Your application should generate a message box when the user does something unexpected, or when your application needs to display an informational message to the user. A message box can freeze your application and require the user to explicitly dismiss the message box before the application may proceed.

To create an instance of the message box widget, use `DwtMessageBox` or `DwtMessageBoxCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The DwtMessageBox and DwtMessageBoxCreate functions create an instance of the message box widget and return its associated widget ID. 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.

The message box widget follows the same rules for geometry management as its superclass the dialog box widget. See the DwtDialogBoxCreate function for more information.

The message box widget follows the same rules for resizing as its superclass the dialog box widget. See the DwtDialogBoxCreate function for more information. The following sections discuss these aspects of the message box widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.3.1 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*.

6.3.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the message box widget is:

- core
- composite
- constraint
- dialog
- message

Based on this class hierarchy, the message box widget inherits attributes from the core, composite, constraint, and dialog box widgets. Note that the message box is a message class widget. Note also that you cannot set the attributes for the composite or constraint widgets; therefore, they are not shown.

Table 6-5 lists the attributes inherited by the message box widget. For descriptions of the core and common attributes, see Chapter 1.

Table 6-5: Attributes Inherited by the Message Box Widget

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

Table 6-5: (continued)

Attribute Name	Data Type	Default
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

Table 6-5: (continued)

Attribute Name	Data Type	Default
DwtNnoResize	Boolean	True (that is, no window manager resize button)
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	NOT SUPPORTED	
DwtNcancelButton	NOT SUPPORTED	

6.3.3 Widget-Specific Attributes

Table 6-6 lists the widget-specific attributes for the message box widget. Descriptions of these attributes follow the table.

Table 6-6: Widget-Specific Attributes for the Message Box Widget

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.

6.4 Creating a Caution Box Widget

A caution box warns the user of the consequences of carrying out an action. It stops application activity and requires the user to provide instructions on how to proceed. The box may contain Yes, No, and Cancel push buttons.

To create an instance of the caution box widget, use DwtCautionBox or DwtCautionBoxCreate. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The `DwtCautionBox` and `DwtCautionBoxCreate` functions create a caution box widget and return its associated widget ID. A caution box warns the user of the consequences of carrying out an action. It stops application activity and requires the user to provide instructions on how to proceed. The box may contain Yes, No, and Cancel push buttons. 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`.

The caution box widget follows the same rules for geometry management as its superclass the dialog box widget. See `DwtDialogBoxCreate` for more information.

The caution box widget follows the same rules for resizing as its superclass the dialog box widget. See `DwtDialogBoxCreate` for more information.

The following sections discuss these aspects of the main window widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.4.1 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*.

6.4.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the caution box widget is:

- core
- composite

- constraint
- dialog
- message

Based on this class hierarchy, the caution box widget inherits attributes from the core, composite, constraint, and dialog box widgets. Note that the caution box is a message class widget. Note also that you cannot set the attributes for the composite or constraint widgets; therefore, they are not shown.

Table 6-7 lists the attributes inherited by the caution box widget. For descriptions of the core and common attributes, see Chapter 1.

Table 6-7: Attributes Inherited by the Caution Box Widget

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

Table 6-7: (continued)

Attribute Name	Data Type	Default
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	

6.4.3 Widget-Specific Attributes

Table 6-8 lists the widget-specific attributes for the caution box widget. Descriptions of these attributes follow the table.

Table 6-8: Widget-Specific Attributes for the Caution Box Widget

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNyesLabel	DwtCompString	"Yes"
DwtNnoLabel	DwtCompString	"No"
DwtNcancelLabel	DwtCompString	"Cancel"

Table 6-8: (continued)

Attribute Name	Data Type	Default
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.

6.5 Creating the Command Window

A command window allows the user to enter commands from the keyboard rather than using the mouse.

To create an instance of the command window widget, use DwtCommandWindow or DwtCommandWindowCreate. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

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

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.

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

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*).

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.

The command window widget follows the same rules for geometry management and resizing as its superclass the dialog box widget, which you can create by calling `DwtDialogBox` or `DwtDialogBoxCreate`.

The following sections discuss these aspects of the command window widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.5.1 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.

6.5.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the command window widget is:

- core
- composite
- constraint
- dialog
- command

Based on this class hierarchy, the command window widget inherits attributes from the core, composite, constraint, and dialog box widgets. Note that you cannot set the attributes for the composite or constraint widgets; therefore, they are not shown.

Table 6-9 lists the attributes inherited by the command window widget. For descriptions of the core and common attributes, see Chapter 1.

Table 6-9: Attributes Inherited by the Command Window Widget

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`

Table 6-9: (continued)

Attribute Name	Data Type	Default
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	

Table 6-9: (continued)

Attribute Name	Data Type	Default
DwtNcancelButton	Widget	NULL
DwtNcancelButton	NOT SUPPORTED	

6.5.3 Widget-Specific Attributes

Table 6-10 lists the widget-specific attributes for the command window widget. Descriptions of these attributes follow the table.

Table 6-10: Widget-Specific Attributes for the Command Window Widget

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.

6.6 Manipulating the Command Line

The XUI Toolkit provides functions with which you can manipulate the command line. This section discusses how to:

- Append a string to the command line
- Write an error message in the history area
- Replace the current command string

To append a passed string to the current command line, use `DwtCommandAppend`.

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

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.

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.

To write an error message in the command window, use `DwtCommandErrorMessage`.

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

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.

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

To replace the current command string with the one passed, use `DwtCommandSet`.

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

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.

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.

6.7 Creating a Selection Box Widget

To create an instance of the selection box widget, use `DwtSelection` or `DwtSelectionCreate`. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

The DwtSelection and DwtSelectionCreate functions create an instance of a selection box widget and return its associated widget ID. 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.

The selection widget follows the same rules for geometry management and resizing as its superclass the dialog box widget. However, the selection widget allows only one child, and it places the child between the list box and the push buttons. The child cannot be a gadget.

See DwtDialogBoxCreate for more information.

The following sections discuss these aspects of the selection box widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.7.1 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.

6.7.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the selection box widget is:

- core

- composite
- constraint
- dialog
- selection

Based on this class hierarchy, the selection widget inherits attributes from the core, composite, constraint, and dialog box widgets. Note that you cannot set the attributes for the composite or constraint widgets; therefore, they are not shown.

Table 6-11 lists the attributes inherited by the selection box widget. For descriptions of the core and common attributes, see Chapter 1.

Table 6-11: Attributes Inherited by the Selection Box Widget

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 th 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

Table 6-11: (continued)

Attribute Name	Data Type	Default
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

6.7.3 Widget-Specific Attributes

Table 6-12 lists the widget-specific attributes for the selection box widget. Descriptions of these attributes follow the table.

Table 6-12: Widget-Specific Attributes for the Selection Box Widget

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.

6.8 Creating a File Selection Box Widget

A file selection box allows the user to specify a file name within your application. To create an instance of the file selection widget, use DwtFileSelection or DwtFileSelectionCreate. 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. After you create an instance of this widget, you can manipulate it using the appropriate X intrinsics functions. A description of each follows:

```
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;
```

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.

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

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*).

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. 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.

The file selection widget follows the same rules for geometry management as its superclass the selection widget. The selection widget follows the same rules for geometry management as its superclass the dialog widget. Consequently, the selection widget allows only one child and it places the child between the list box and the push buttons. The child cannot be a gadget. For more information on the geometry management associated with the dialog widget, see Section 5.1.2.

The file selection widget follows the same rules for resizing as its superclass the selection widget. The selection widget follows the same rules for resizing as its superclass the dialog widget. For more information on the resizing associated with the dialog widget, see the `DwtNresize` attribute associated with `DwtDialogBoxCreate`.

The following sections discuss these aspects of the file selection box widget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

6.8.1 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.

6.8.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the file selection box widget is:

- core
- composite
- constraint
- dialog
- selection
- file selection

Based on this class hierarchy, the file selection widget inherits attributes from the core, composite, constraint, dialog, and selection widgets. Note that you cannot set the attributes for the composite or constraint widgets; therefore, they are not shown.

Table 6-13 lists the attributes inherited by the file selection box widget. For descriptions of the core and common attributes, see Chapter 1.

Table 6-13: Attributes Inherited by the File Selection Box Widget

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.

Table 6-13: (continued)

Attribute Name	Data Type	Default
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 an 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

Table 6-13: (continued)

Attribute Name	Data Type	Default
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"

6.8.3 Widget-Specific Attributes

Table 6-14 lists the widget-specific attributes for the file selection box widget. Descriptions of these attributes follow the table.

Table 6-14: Widget-Specific Attributes for the File Selection Box Widget

Attribute Name	Data Type	Default
DwtNfilterLabel	DwtCompString	"File filter"
DwtNapplyLabel	DwtCompString	"Filter"

Table 6-14: (continued)

Attribute Name	Data Type	Default
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.

6.9 Initiating a Search with a Directory Mask Option

To initiate a search with a directory mask option, use DwtFileSelectionDoSearch.

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

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.

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.

Gadget Functions 7

The XUI Toolkit provides gadgets, which are reduced functionality widgets. The main difference between a gadget and a widget is that a gadget does not have an associated X window. This chapter discusses the following:

- Classes associated with gadgets
- Advantages of using gadgets
- X intrinsics and convenience functions used with gadgets

In addition, the chapter explains the functions you use to create instances of the following:

- Label gadget
- Separator gadget
- Push button gadget
- Toggle button gadget
- Pull-down menu entry gadget

7.1 Classes Associated with Gadgets

Table 7-1 lists each gadget and its associated class and possible parent widgets.

Table 7-1: Gadget Classes and Parents

Gadget	Class	Parent
`DwtLabelGadgetCreate`	labelgadgetclass	The parent widget must be a menu class widget (Menu, Menu Bar, Pull-Down Menu, Pop-Up Menu, Radio Box) or a dialog box class widget.

Table 7-1: (continued)

Gadget	Class	Parent
`DwtSeparatorGadgetCreate`	separatorgadgetclass	Same as for `DwtLabelGadgetCrea`
`DwtPushButtonGadgetCreate`	pushbuttongadgetclass	Same as for `DwtLabelGadgetCrea`
`DwtToggleButtonGadgetCreate`	togglebuttongadgetclass	Same as for `DwtLabelGadgetCrea`

Note that because a parent widget must do work for the gadget and a gadget does not have an associated X window, you cannot place gadgets within a composite widget.

7.2 Advantages of Using Gadgets

The following are advantages of using gadget widgets:

- Reduced creation time — Gadgets have far less resources than widgets
- Reduced memory usage — Gadget instance records are much smaller than widget instance records
- Reduced window count — Gadgets do not create an associated X window
- Increased drawing speed — A side-effect of not having a separate window

7.3 X Intrinsics and Convenience Functions Used with Gadgets

The following functions are supported by `DwtToggleButtonGadgetCreate`:

- `DwtToggleButtonGetState`
- `DwtToggleButtonSetState`

The following X intrinsics functions are supported by all the gadgets for calling by applications:

- `XtDestroyWidget`
- `XtSetValues`

- XtGetValues
- XtClass
- XtIsSubclass
- XtSuperclass
- XtCheckSubclass
- XtAddCallback
- XtAddCallbacks
- XtRemoveCallback
- XtRemoveCallbacks
- XtHasCallbacks
- XtSetSensitive
- XtConvert

7.4 Creating a Label Gadget

To create an instance of the label gadget, use DwtLabelGadgetCreate. For DwtLabelGadgetCreate, just like the usual low-level functions, you specify a list of attribute name/value pairs that represent all the possible label gadget attributes. A description of this function follows:

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

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*).

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.

Because a label gadget is not a subclass of composite, children are not supported.

The label gadget does nothing on a resize by its parents.

The following sections discuss these aspects of the label gadget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

7.4.1 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*.

7.4.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the label gadget is:

- object
- rectangle
- label gadget

Table 7-2 lists the attributes inherited by the label gadget.

Table 7-2: Attributes Inherited by the Label Gadget

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

7.4.3 Widget-Specific Attributes

Table 7-3 lists the widget-specific attributes of the label gadget. Following the table are descriptions of each attribute.

Table 7-3: Widget-Specific Attributes for DwtLabelGadgetCreate

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.

7.5 Creating a Push Button Gadget

To create an instance of the push button gadget, use `DwtPushButtonGadgetCreate`. For `DwtPushButtonGadgetCreate`, just like the usual low-level functions, you specify a list of attribute name/value pairs that represent all the possible push button gadget attributes. A description of this function follows:

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

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*).

The `DwtPushButtonGadgetCreate` function creates an instance of the push button gadget and returns its associated gadget ID. Because drawing information such as font and color are those of the closest antecedent, the sizing is affected by the font and label. See `DwtLabelGadgetCreate`.

Because a push button gadget is not a subclass of composite, children are not supported. The push button gadget widget does nothing on a resize by its parents.

The following sections discuss these aspects of the push button gadget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

7.5.1 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*.

7.5.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the push button gadget is:

- object
- rectangle
- label gadget
- push button gadget

Table 7-4 lists the attributes inherited by the push button gadget.

Table 7-4: Attributes Inherited by the Push Button Gadget

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

Table 7-4: (continued)

Attribute Name	Data Type	Default
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

7.5.3 Widget-Specific Attributes

Table 7-5 lists the widget-specific attributes of the push button gadget. Following the table are descriptions of each attribute.

Table 7-5: Widget-Specific Attributes for the Push Button Gadget

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.

7.6 Creating a Separator Gadget

To create an instance of the separator gadget, use DwtSeparatorGadgetCreate. For DwtSeparatorGadgetCreate, just like the usual low-level functions, you specify a list of attribute name/value pairs that represent all the possible separator gadget attributes. A description of this function follows:

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

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*).

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.

The separator gadget widget does nothing on a resize by its parents. Because a separator gadget is not a subclass of composite, children are not supported.

The following sections discuss these aspects of the separator gadget:

- Widget class hierarchy and inherited attributes

- Widget-specific attributes

7.6.1 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the separator gadget widget is:

- object
- rectangle
- separator gadget

Table 7-6 lists the attributes inherited by the separator gadget.

Table 7-6: Attributes Inherited by the Separator Gadget

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

7.6.2 Widget-Specific Attributes

Table 7-7 lists the widget-specific attribute of the separator gadget. Following the table is a description of the attribute.

Table 7-7: Widget-Specific Attribute for the Separator Gadget

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.

7.7 Creating a Toggle Button Gadget

To create an instance of the toggle button gadget, use DwtToggleButtonGadgetCreate. For DwtToggleButtonGadgetCreate, just like the usual low-level functions, you specify a list of attribute name/value pairs that represent all the possible toggle button gadget attributes. A description of this function follows:

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

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*).

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.

The sizing is affected by the font and label. See DwtLabelGadgetCreate for more information. The indicator size is based on the height of the toggle button. The indicator width is equal to the

indicator height.

The following sections discuss these aspects of the toggle button gadget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

7.7.1 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).

7.7.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the toggle button gadget widget is:

- object
- rectangle
- label gadget
- toggle button gadget

Table 7-8 lists the attributes inherited by the toggle button gadget.

Table 7-8: Attributes Inherited by the Toggle Button Gadget

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

7.7.3 Widget-Specific Attributes

Table 7-9 lists the widget-specific attributes of the toggle button gadget. Following the table are descriptions of each attribute.

Table 7-9: Widget-Specific Attributes for the Toggle Button Gadget

Attribute Name	Data Type	Default
DwtNshape	unsigned char	DwtRectangular
DwtNvalue	Boolean	False
DwtNvisibleWhenOff	Boolean	True
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNbuttonAccelerator	char *	NULL
DwtNacceleratorText	DwtCompString	NULL

Table 7-9: (continued)

Attribute Name	Data Type	Default
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.	

7.8 Creating a Pull-Down Menu Entry Gadget

To create an instance of the pull-down menu entry gadget, use DwtPullEntryGadgetCreate. For DwtPullEntryGadgetCreate, just like the usual low-level functions, you specify a list of attribute name/value pairs that represent all the possible pull-down menu entry gadget attributes. A description of this function follows:

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

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*).

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. See the section on geometry management and resizing for the `DwtPullDownMenuEntry` function for more information. The following sections discuss these aspects of the pull-down menu entry gadget:

- Callback information
- Widget class hierarchy and inherited attributes
- Widget-specific attributes

7.8.1 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*.

7.8.2 Widget Class Hierarchy and Inherited Attributes

The widget class hierarchy for the pull-down menu entry gadget is:

- object
- rectangle
- label gadget
- pull-down menu entry gadget

Table 7-10 lists the attributes inherited by the pull down menu entry gadget.

Table 7-10: Attributes Inherited by the Pull-Down Menu Entry Gadget

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

Table 7-10: (continued)

Attribute Name	Data Type	Default
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

7.8.3 Widget-Specific Attributes

Table 7-11 lists the widget-specific attributes of the pull-down menu entry gadget. Following the table are descriptions of each attribute.

Table 7-11: Widget-Specific Attributes for the Pull-Down Menu Entry Gadget

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`.

PART THREE: OTHER TOOLKIT FUNCTIONS

Cut and Paste Functions 8

An application can interface to the XUI Toolkit clipboard through calls to the cut and paste functions. For more information, see the *XUI Style Guide*.

This chapter begins with an introduction to the cut and paste functions followed by a brief discussion of the formats. The rest of the chapter discusses the functions you can use to:

- Set up storage and data structures
- Indicate that the application no longer wants to supply a data item
- Cancel a copy to clipboard
- Lock the clipboard
- Unlock the clipboard
- Retrieve a data item from the clipboard
- Copy a data item to the clipboard
- Place data in the clipboard
- Return the number of data item formats
- Return a specified format name
- Return the length of the stored data
- Return a list of data ID/private ID pairs
- Copy a data item passed by name
- Delete the last item placed on the clipboard
- Register data length for non-Inter-Client Communications Conventions Manual formats

8.1 Introduction to the Cut and Paste Functions

If your application needs to give the user maximum access to the clipboard, the following menu items should be included in the Edit menu:

- Cut

  When the user chooses this item, the application calls `DwtBeginCopyToClipboard`, `DwtCopyToClipboard`, and `DwtEndCopyToClipboard` to copy the data in whatever formats it

desires. The application should then delete the cut items from the user's display.

- Copy

  The same as Cut except the items are not deleted.

- Paste

  When the user chooses this item, the application calls `DwtCopyFromClipboard` to obtain the data in some format. The application then allows the user to place the data on the display.

The user can perform the following actions on an application by using the clipboard:

- Undo Cut

  The application redraws the deleted items. The application then calls `DwtUndoCopyToClipboard` to delete the items from the clipboard.

- Redo Cut

  The application copies the items to the clipboard.

- Undo Copy

  The application calls `DwtUndoCopyToClipboard`.

- Redo Copy

  The application copies the items to the clipboard.

- Undo Paste

  The application saves the pasted items for a possible later Redo Paste.

- Redo Paste

  The application saves the pasted items for a possible later Undo Paste.

The clipboard is not involved with an Undo Paste or a Redo Paste. This is because between the Paste and the Undo Paste or between the Undo Paste and the Redo Paste, the clipboard might be changed by the user either directly or indirectly.

Copying a large piece of data to the clipboard can take time. It is possible that, once copied, no application will ever request that data. The XUI Toolkit provides a mechanism so that an application does not need to actually pass data to the clipboard until the data has been requested by some application. Instead, the application passes format and length information to the clipboard functions, along with a widget ID and a callback function address. The widget ID is needed for communications between the clipboard functions in the application that owns the data and the clipboard functions in the application that requests the data. Your callback functions are responsible for copying the actual data to the clipboard (via

DwtReCopyToClipboard). The callback function is also called if the data item is removed from the clipboard, and the actual data is therefore no longer needed.

For more information on passing data by name, see DwtBeginCopyToClipboard, DwtCopyToClipboard, and DwtReCopyToClipboard.

8.2 ICCCM Compliant Functions and ICCCM Formats

Four new cut and paste functions have been added to ensure compliance with the Inter-Client Communications Conventions Manual (ICCCM). The ICCCM manual defines conventions for using the global selection mechanism that allows compliant clients to communicate with each other. These new functions include:

- DwtStartCopyToClipboard
- DwtStartCopyFromClipboard
- DwtEndCopyFromClipboard
- DwtClipboardRegisterFormat

Several of the cut and paste functions use the *format_name* argument, which specifies the format of the data stored on the clipboard. The formats specified by the ICCCM are 8-bit, 16-bit, and 32-bit.

It is recommended that applications use the ICCCM formats. If an application uses a non-ICCCM format that is not an 8-bit format, it must register the format with the cut and paste functions using DwtClipboardRegisterFormat. Failure to do so may result in unexpected results when trying to cut and paste between clients running on different platforms.

Table 8-1 lists the ICCCM format names, format lengths, and a description of the formats.

Table 8-1: Data Format Names

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

Table 8-1: (continued)

Format Name	Format Length	Description
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*.

8.3 Setting Up Storage and Data Structures

To set up storage and data structures to receive clipboard data, use `DwtBeginCopyToClipboard` or `DwtStartCopyFromClipboard`. Because it complies with the ICCCM conventions, use of `DwtStartCopyToClipboard` is recommended over `DwtBeginCopyToClipboard`. Each of these functions is discussed in the following sections.

8.3.1 Using DwtStartCopyFromClipboard to Set Up Storage and Data Structures

To comply with ICCCM conventions when setting up storage and data structures, use `DwtStartCopyToClipboard`.

```
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;
```

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`.

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.

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.

8.3.2 Using DwtBeginCopyToClipboard to Set Up Storage and Data Structures

To set up storage and data structures, use DwtBeginCopyToClipboard.

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

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.

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`.

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.

8.4 Indicating That the Application No Longer Wants to Supply a Data Item

To indicate that the application is no longer willing to supply a data item to the clipboard, use `DwtCancelCopyFormat`.

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

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`.

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.

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.

8.5 Canceling a Copy to Clipboard

To cancel the copy to clipboard that is in progress, use `DwtCancelCopyToClipboard`

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

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.

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.

8.6 Locking the Clipboard

To lock the clipboard from access by other applications, use DwtClipboardLock.

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

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.

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.

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.

8.7 Unlocking the Clipboard

To unlock the clipboard, use DwtClipboardUnlock.

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

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.

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.

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.

8.8 Retrieving a Data Item from the Clipboard

You can use `DwtCopyFromClipboard` to obtain the current next paste data item from clipboard storage. Because the matching functions `DwtStartCopyFromClipboard` and `DwtEndCopyFromClipboard` comply with the ICCCM conventions, their use is recommended over `DwtCopyFromClipboard`. Each of these functions is discussed in the following sections.

8.8.1 Using DwtStartCopyFromClipboard and DwtEndCopyFromClipboard

To indicate that the application is ready to start copying data from the clipboard, use `DwtStartCopyFromClipboard`.

```
int DwtStartCopyFromClipboard(display, window, time)
     Display *display;
     Window window;
     Time time;
```

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.

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.

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.

To indicate that the application is ready to start copying data from the clipboard, use DwtEndCopyFromClipboard.

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

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.

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.

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.

8.8.2 Using DwtCopyFromClipboard

To obtain the current next paste data item from clipboard storage, use `DwtCopyFromClipboard`.

```
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;
```

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.

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

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.

8.9 Copying a Data Item to the Clipboard

To copy a data item to clipboard storage, use DwtCopyToClipboard.

```
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;
```

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.

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.

For information on the callback function, see the callback argument description for `DwtBeginCopyToClipboard`.

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.

8.10 Placing Data in the Clipboard

To lock the clipboard, to place data in the clipboard data structure, and to unlock the clipboard, use `DwtEndCopyToClipboard`.

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

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`.

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`.

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.

8.11 Returning the Number of Data Item Formats

To return the maximum length for all data item formats, use `DwtInquireNextPasteCount`.

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

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.

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.

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.

8.12 Returning a Specified Format Name

To obtain the format name for the next paste data item in the clipboard, use `DwtInquireNextPasteFormat`.

```
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;
```

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.

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. 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.

8.13 Returning the Length of the Stored Data

To obtain the length of the data stored under a specified format, use `DwtInquireNextPasteLength`.

```
int DwtInquireNextPasteLength(display, window,
                              format_name, length)
    Display *display;
    Window window;
    char *format_name;
    unsigned long *length;
```

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.

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.

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.

8.14 Returning a List of Data ID/Private ID Pairs

To obtain a format name's list of data ID or private ID pairs, use `DwtListPendingItems`.

```
int DwtListPendingItems(display, window, format_name,
                        item_list, count)
    Display *display;
    Window window;
    char *format_name;
    DwtClipboardPendingList *item_list;
    unsigned long *count;
```

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.

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.

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.

8.15 Copying a Data Item Passed by Name

To copy a data item to the clipboard, use `DwtReCopyToClipboard`.

```
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;
```

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.

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.

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.

8.16 Deleting the Last Item Placed on the Clipboard

To delete the last item placed on the clipboard, use `DwtUndoCopyToClipboard`.

```
int DwtUndoCopyToClipboard(display, window)
     Display *display;
     Window window;
```

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.

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.

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.

8.17 Register Data Length for Non-ICCCM Formats

To register the data length for non-ICCCM formats, use `DwtClipboardRegisterFormat`.

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

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.

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.

See Table 8-1 for the formats defined by the ICCCM conventions.

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.

Compound String Functions 9

The XUI Toolkit provides a set of compound string functions that enable the creation and manipulation of compound strings and font lists. A compound string is a sequence of segments. Each segment consists of a natural language identifier, a text direction identifier, rendition information, a character set identifier, and a counted text string.

A font list is an array of font structures indexed by the character set identifier. A character set identifier is a constant from the file `/usr/include/cda_def.h`. This chapter discusses the compound string functions you can use to:

- Create a font list and font list entries
- Create compound strings
- Compare and manipulate compound strings
- Free a compound string context structure

9.1 Creating a Font List and Font List Entries

The functions discussed in this section allow you to create a new font list and to add entries to this list.

To create a new font list, use `DwtCreateFontList`.

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

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`.

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.

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

To add an entry to a font list, use DwtAddFontList.

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

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.

The DwtAddFontList function adds an entry to a font list.

This function returns the new font list.

9.2 Creating a Compound String

The functions discussed in this section allow you to create a compound string and to create a compound-string for the LATIN1 character set.

To create a compound-string, use DwtCSString.

```
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;
```

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.

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`.

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

To create a compound-string using a simpler interface than the one used for `DwtCSString`, call `DwtString`.

```
DwtCompString DwtString(text, charset, direction_r_to_l)
      char *text;
      unsigned long charset;
      char direction_r_to_l;
```

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).

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`.

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

To create a compound-string for the LATIN1 character set, use `DwtLatin1String`.

```
DwtCompString DwtLatin1String(text)
      char *text;
```

text Specifies the text string to be converted to a compound-string.

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.

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`.

9.3 Comparing and Manipulating Compound Strings

The functions discussed in this chapter allow your application to:

- Determine if two strings are identical
- Determine if a compound-string is empty
- Append a copy of a string to another string
- Copy a string to the output string
- Return number of bytes in string1
- Work with segments

9.3.1 Determining If Two Compound Strings Are Identical

To determine if two compound-strings are identical, use `DwtCSbytecmp`.

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

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

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

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.

9.3.2 Determining If a Compound String Is Empty

To determine if a compound-string is empty, use `DwtCSempty`.

```
int DwtCSempty (compound_string)
      DwtCompString compound_string;
```

*compound_string*Specifies the compound-string.

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`.

9.3.3 Appending a Copy of a String to Another String

To append a copy of a compound-string to another compound-string, use `DwtCStrcat` or `DwtCStrncat`.

```
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;
```

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.

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.

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

9.3.4 Copying a String to the Output String

To copy a compound-string to the output string, use DwtCStrcpy or DwtCStrncpy.

```
DwtCompString DwtCStrcpy (compound_string1)
     DwtCompString compound_string1;

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

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.

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.

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

9.3.5 Returning the Number of Bytes in String1

To obtain the length of *compound_string1*, use DwtCStrlen.

```
int DwtCStrlen(compound_string1)
    DwtCompString compound_string1;
```

compound_string1
: Specifies a compound-string whose length is determined.

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.

9.3.6 Working with Segments

The functions discussed in this section allow you to obtain information about the next segment, to obtain the initialized context of the compound string, and to initialize a compound string context structure. To obtain information about the next segment in a compound string, use `DwtGetNextSegment`.

```
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;
```

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.

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`.

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.

To obtain the initialized context of the compound string, use DwtInitGetSegment.

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

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

*compound_string*Specifies the compound-string.

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.

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.

To initialize a compound string context structure, use DwtStringInitContext.

```
Boolean DwtStringInitContext (context, compound_string)
     DwtCompStringContext context;
     DwtCompString compound_string;
```

context Specifies the compound-string context structure initialized by `DwtStringInitContext`.

*compound_string*Specifies the compound-string.

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.

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.

9.4 Freeing a Compound String Context Structure

To free a compound string context structure, use `DwtStringFreeContext`.

```
void DwtStringFreeContext (context)
     DwtCompStringContext *context;
```

context Specifies the compound-string context structure initialized by `DwtStringInitContext`.

The `DwtStringFreeContext` function frees the compound-string context structure returned by `DwtStringInitContext`. When your application has finished with the context, it should call `DwtStringFreeContext`.

Convenience Functions 10

This chapter discusses the following types of convenience functions:

- Functions that allow you to display messages
- A function that allows you to write upward-compatible applications and widgets
- A function that allows applications to simulate push button activation
- A function that returns user data associated with the widget

10.1 Functions That Display Messages

The message functions allow messages to be formatted using the $FAO utility and to be displayed in a message box. Messages such as those from the operating system appear in a message box. The messages themselves are either stored in a VMS message file or supplied by the application as compound-strings.

The message box can be either modal or modeless. The title bar identifies the message box with the title Message and contains a push-to-back icon. The message box contains a message box icon in the upper-left corner, an Acknowledged push button on the bottom, and the text of the message in the middle. The lines of the message are separated by <CR><LF> pairs or "!/" in $FAO. Multiple messages are separated by blank lines.

The user clicks on the Acknowledged push button to erase the message. To receive help on the message, the user presses the Help key while the message box has input focus.

In addition to the standard $FAO system service flags, the compound string message function (`DwtDisplayCSMessage`) accepts the flag "!CS". When used, this flag accepts a compound-string itself.

ULTRIX applications can make use of the FAO string substitution utility. Messages defined in the Resource Manager (DRM) database can be supplied to the appropriate message function to be formatted and displayed.

The following sections discuss the functions you can use to:

- Accept and display a VMS message
- Display a compound-string message

10.1.1 Accepting and Displaying a VMS Message

To accept and display a VMS message, use `DwtDisplayVmsMessage`.

```
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;
```

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.

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

This parameter is a NULL-terminated character string.

10.1.2 Displaying a Compound String Message

To display a compound-string message, use `DwtDisplayCSMessage`.

```
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;
```

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.

The DwtDisplayCSMessage function accepts an array of compound-strings, formats them, and creates a message box. Upon completion, DwtDisplayCSMessage returns to the calling program the ID of the created message box widget.

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

10.2 Function That Allows Writing Upward-Compatible Applications and Widgets

The widget instance records defined by the XUI Toolkit are not inherently upward compatible. That is, applications and widgets built outside of the toolkit cannot use hard-coded offsets into the widget instance records and expect to run without recompilation as subsequent releases of the XUI Toolkit occur. A method of writing upward-compatible applications and widgets has been defined.

To allocate offset records, use DwtResolvePartOffsets.

```
void DwtResolvePartOffsets (widget_class, offset)
     WidgetClass widget_class;
     DwtOffsetPtr *offset;
```

widget_class Specifies the widget class pointer for the created widget.

offset Specifies the offset record.

Applications and widgets must never read or write another widget's fields directly. There is one exception to this: as geometry manager, a widget's parent may read and write the widget's geometry fields. A widget not guaranteed to be recompiled with the intrinsics must use an "offset record" to access the field within its instance record.

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 *)
```

10.3 Function That Allows Applications to Simulate Push Button Activation

To allow your application to simulate push button activations, use `DwtActivateWidget`.

```
void DwtActivateWidget (widget)
     Widget widget;
```

widget Specifies a pointer to the widget data structure.

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`.

10.4 Function That Returns User Data Associated with the Widget

To return the user data associated with the widget, use `DwtGetUserData`.

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

widget Specifies a pointer to the widget data structure.

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

Widget Attributes A

This appendix provides tables that list the attributes supported by each widget. Each table provides the attribute name, data type, and default. For your convenience, the widgets are listed in alphabetical order.

A.1 Attached Dialog Box and Attached Dialog Box Pop-Up Widgets

Table A-1: Attributes Inherited by the Attached Dialog Box and Attached Dialog Box Pop-Up Widgets

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
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

Table A-1: (continued)

Attribute Name	Data Type	Default
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 DwtNdefaultVerticalOffs However, if DwtNadbTopAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value specified with DwtNdefaultVerticalOffs

Table A-1: (continued)

Attribute Name	Data Type	Default
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 DwtNdefaultHorizontalOffset. However, if DwtNadbLeftAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value of DwtNdefaultHorizontalOffset.
DwtNadbRightOffset	int	The value specified with DwtNdefaultHorizontalOffset. However, if DwtNadbRightAttachment is DwtAttachPosition or DwtAttachSelf, the default is one-half the value specified with DwtNdefaultHorizontalOffset.
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

Table A-1: (continued)

Attribute Name	Data Type	Default
DwtNmarginWidth	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreat the default is 3 pixels.
DwtNmarginHeight	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreat the default is 3 pixels.
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeGrowOnly
DwtNgrabKeySyms	KeySym	The default array contains the T key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT()\ Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()"

Table A-2: Widget-Specific Attributes for the Attached Dialog Box and Attached Dialog Box Pop-Up Widgets

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

A.2 Caution Box Widget

Table A-3: Attributes Inherited by the Caution Box Widget

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

Table A-3: (continued)

Attribute Name	Data Type	Default
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	

Table A-4: Widget-Specific Attributes for the Caution Box Widget

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

A.3 Color Mix Widget

Table A-5: Attributes Inherited by the Color Mix Widget

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

Table A-5: (continued)

Attribute Name	Data Type	Default
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 Ta key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT()\r Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()";

Table A-6: Widget-Specific Attributes for the Color Mix Widget

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.

Table A-6: (continued)

Attribute Name	Data Type	Default
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.

Table A-6: (continued)

Attribute Name	Data Type	Default
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.

A.4 Command Window Widget

Table A-7: Attributes Inherited by the Command Window Widget

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

Table A-7: (continued)

Attribute Name	Data Type	Default
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 windo 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	

Table A-8: Widget-Specific Attributes for the Command Window Widget

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

A.5 Compound String Text Widget

Table A-9: Attributes Inherited by the Compound String Text Widget

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

Table A-10: Widget-Specific Attributes for the Compound String Text Widget

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	DwtDirectionRightDow
DwtNtextPath	int	Left to right
DwtNeditingPath	int	Left to right
DwtNbidirectionalCursor	Boolean	False
DwtNnofontCallback	DwtCallbackPtr	NULL

A.6 Dialog Box and Pop-Up Dialog Box Widgets

Table A-11: Attributes Inherited by the Dialog Box and Pop-Up Dialog Box Widgets

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

Table A-12: Widget-Specific Attributes for the Dialog Box and Pop-Up Dialog Box Widgets

Attribute Name	Data Type	Default
DwtNforeground	Pixel	Default foreground color
DwtNhighlight	Pixel	Default foreground color
DwtNhighlightPixmap	Pixmap	NULL

Table A-12: (continued)

Attribute Name	Data Type	Default
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 DwtDialogBoxPopupCreat the default is DwtModeless.
DwtNfocusCallback	DwtCallbackPtr	NULL
DwtNtextMergeTranslations	XtTranslations	NULL
DwtNmarginWidth	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreat the default is 3 pixels.
DwtNmarginHeight	Dimension	For DwtDialogBoxCreate, the default is One pixel. For DwtDialogBoxPopupCreat the default is 3 pixels.
DwtNdefaultPosition	Boolean	False
DwtNchildOverlap	Boolean	True
DwtNresize	unsigned char	DwtResizeGrowOnly
DwtNgrabKeySyms	KeySym	The default array contains the T key symbol.
DwtNgrabMergeTranslations	XtTranslations	The default syntax is: "~Shift<KeyPress>0xff09: DWTDIMOVEFOCUSNEXT() Shift<KeyPress>0xff09: DWTDIMOVEFOCUSPREV()"

Table A-13: 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

Table A-13: (continued)

Attribute Name	Data Type	Default
		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

A.7 File Selection Widget

Table A-14: Attributes Inherited by the File Selection Widget

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

Table A-14: (continued)

Attribute Name	Data Type	Default
DwtNcolormap	Colormap	Default color map
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive ai 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 bo:
DwtNautoUnmanage	Boolean	True
DwtNdefaultButton	Widget	NULL
DwtNcancelButton	Widget	NULL

Table A-14: (continued)

Attribute Name	Data Type	Default
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"

Table A-15: Widget-Specific Attributes for the File Selection Widget

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

A.8 Help Widget

Table A-16: Attributes Inherited by the Help Widget

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

Table A-16: (continued)

Attribute Name	Data Type	Default
DwtNdirectionRToL	unsigned char	DwtDirectionRightDown
DwtNfont	DwtFontList	The default XUI Toolkit font
DwtNhelpCallback	DwtCallbackPtr	NULL

Table A-17: Widget-Specific Attributes for the Help Widget

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"

Table A-17: (continued)

Attribute Name	Data Type	Default
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"

A.9 Label Widget

Table A-18: Attributes Inherited by the Label Widget

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

Table A-19: Widget-Specific Attributes for the Label Widget

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

A.10 Label Gadget

Table A-20: Attributes Inherited by the Label Gadget

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

Table A-20: (continued)

Table A-21: Widget-Specific Attributes for the the Label Gadget

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNdirectionRToL	Boolean	False
DwtNhelpCallback	DwtCallbackPtr	NULL

A.11 List Box Widget

Table A-22: Attributes Inherited by the List Box Widget

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

Table A-22: (continued)

Attribute Name	Data Type	Default
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	DwtDirectionRightDow
DwtNfont	NOT SUPPORTED	
DwtNhelpCallback	NOT SUPPORTED	
Scroll Window Attributes		
DwtNhorizontalScrollBar	Widget	NULL
DwtNverticalScrollBar	Widget	NULL
DwtNworkWindow	Widget	NULL
DwtNshownValueAutomaticHoriz	Boolean	True
DwtNshownValueAutomaticVert	Boolean	False

Table A-23: Widget-Specific Attributes for the List Box Widget

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.

Table A-23: (continued)

Attribute Name	Data Type	Default
DwtNsingleSelection	Boolean	True
DwtNresize	Boolean	True
DwtNhorizontal	Boolean	False
DwtNsingleCallback	DwtCallbackPtr	NULL
DwtNsingleConfirmCallback	DwtCallbackPtr	NULL
DwtNextendCallback	DwtCallbackPtr	NULL
DwtNextendConfirmCallback	DwtCallbackPtr	NULL

A.12 Main Window Widget

Table A-24: Attributes Inherited by the Main Window Widget

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

Table A-24: (continued)

Attribute Name	Data Type	Default
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

Table A-25: Widget-Specific Attributes for the Main Window Widget

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

A.13 Menu Bar Widget

Table A-26: Attributes Inherited by the Menu Bar Widget

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

Table A-26: (continued)

Attribute Name	Data Type	Default
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

Table A-26: (continued)

Attribute Name	Data Type	Default
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

A.14 Menu, Pull-Down Menu, and Pop-Up Menu Widgets

Table A-27: 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.

Table A-27: (continued)

Attribute Name	Data Type	Default
		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

Table A-28: 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

Table A-28: (continued)

Attribute Name	Data Type	Default
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

Table A-29: Widget-Specific Attributes for the Pull-Down Menu and Pop-Up Menu Widgets

Attribute Name	Data Type	Default
DwtNmapCallback	DwtCallbackPtr	NULL
DwtNunmapCallback	DwtCallbackPtr	NULL

Table A-30: Widget-Specific Attributes for the Menu Widget

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

A.15 Message Box Widget

Table A-31: Attributes Inherited by the Message Box Widget

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

Table A-31: (continued)

Attribute Name	Data Type	Default
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	

Table A-32: Widget-Specific Attributes for the Message Box Widget

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.

A.16 Option Menu Widget

Table A-33: Attributes Inherited by the Option 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	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

Table A-33: (continued)

Attribute Name	Data Type	Default
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

Table A-34: Widget-Specific Attributes for the Option Menu Widget

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNsubMenuId	Widget	Zero

A.17 Pull-Down Menu Entry Widget

Table A-35: Attributes Inherited by the Pull Down Menu Entry Widget

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

Table A-35: (continued)

Attribute Name	Data Type	Default
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

Table A-36: Widget-Specific Attributes for the Pull Down Menu Entry Widget

Attribute Name	Data Type	Default
DwtNsubMenuId	Widget	NULL
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNpullingCallback	DwtCallbackPtr	NULL
DwtNhotSpotPixmap	Pixmap	NULL

A.18 Pull-Down Menu Entry Gadget

Table A-37: Attributes Inherited by the Pull-Down Menu Entry Gadget

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

Table A-38: Widget-Specific Attributes for the the Pull-Down Menu Entry Gadget

Attribute Name	Data Type	Default
DwtNsubMenuId	Widget	NULL
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNpullingCallback	DwtCallbackPtr	NULL

A.19 Push Button Widget

Table A-39: Attributes Inherited by the Push Button Widget

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

Table A-39: (continued)

Attribute Name	Data Type	Default
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

Table A-40: Widget-Specific Attributes for the Push Button Widget

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

A.20 Push Button Gadget

Table A-41: Attributes Inherited by the Push Button Gadget

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

Table A-42: Widget-Specific Attributes for the the Push Button Gadget

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	NULL
DwtNactivateCallback	DwtCallbackPtr	NULL
DwtNacceleratorText	DwtCompString	NULL
DwtNbuttonAccelerator	char *	NULL

A.21 Radio Box Widget

Table A-43: Attributes Inherited by the Radio Box Widget

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

Table A-43: (continued)

Attribute Name	Data Type	Default
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

A.22 Scale Widget

Table A-44: Attributes Inherited by the Scale Widget

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

Table A-45: Widget-Specific Attributes for the Scale Widget

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

A.23 Scroll Bar Widget

Table A-46: Attributes Inherited by the Scroll Bar Widget

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

Table A-46: (continued)

Attribute Name	Data Type	Default
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

Table A-47: Widget-Specific Attributes for the Scroll Bar Widget

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

Table A-47: (continued)

Attribute Name	Data Type	Default
DwtNtoTopCallback	DwtCallbackPtr	NULL
DwtNtoBottomCallback	DwtCallbackPtr	NULL
DwtNdragCallback	DwtCallbackPtr	NULL
DwtNshowArrows	Boolean	True

A.24 Scroll Window Widget

Table A-48: Attributes Inherited by the Scroll Window Widget

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

Table A-48: (continued)

Attribute Name	Data Type	Default
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	

Table A-49: Widget-Specific Attributes for the Scroll Window Widget

Attribute Name	Data Type	Default
DwtNhorizontalScrollBar	Widget	NULL
DwtNverticalScrollBar	Widget	NULL
DwtNworkWindow	Widget	NULL
DwtNshownValueAutomaticHoriz	Boolean	True
DwtNshownValueAutomaticVert	Boolean	True

A.25 Selection Widget

Table A-50: Attributes Inherited by the Selection Widget

Attribute Name	Data Type	Default
Core Attributes		
DwtNx	Position	Centered in the parent window
DwtNy	Position	Centered in the parent window

Table A-50: (continued)

Attribute Name	Data Type	Default
DwtNwidth	Dimension	The width of the list box, plus th 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 anc 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

Table A-50: (continued)

Attribute Name	Data Type	Default
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

Table A-51: Widget-Specific Attributes for the Selection Widget

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"

A.26 Separator Widget

Table A-52: Attributes Inherited by the Separator Widget

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		

Table A-52: (continued)

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

Table A-53: Widget-Specific Attributes for the Separator Widget

Attribute Name	Data Type	Default
DwtNorientation	unsigned char	DwtOrientationHorizontal

A.27 Separator Gadget

Table A-54: Attributes Inherited by the Separator Gadget

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

Table A-54: (continued)

Attribute Name	Data Type	Default
DwtNborderWidth	Dimension	zero
DwtNsensitive	Boolean	True
DwtNancestorSensitive	Boolean	The bitwise AND of the parent widget's DwtNsensitive and DwtNancestorSensitive attributes

Table A-55: Widget-Specific Attributes for the the Separator Gadget

Attribute Name	Data Type	Default
DwtNorientation	unsigned char	DwtOrientationHorizontal

A.28 Simple Text Widget

Table A-56: Attributes Inherited by the Simple Text Widget

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

Table A-56: (continued)

Attribute Name	Data Type	Default
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

Table A-57: Widget-Specific Attributes for the Simple Text Widget

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

Table A-57: (continued)

Attribute Name	Data Type	Default
DwtNfont	DwtFontList	The current server font list.
DwtNblinkRate	int	500 milliseconds
DwtNscrollLeftSide	Boolean	False
DwtNhalfBorder	Boolean	True
DwtNpendingDelete	Boolean	True
DwtNuserData	Opaque *	NULL

A.29 Toggle Button Widget

Table A-58: Attributes Inherited by the Toggle Button Widget

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

Table A-58: (continued)

Attribute Name	Data Type	Default
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

Table A-59: Widget-Specific Attributes for the Toggle Button Widget

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

A.30 Toggle Button Gadget

Table A-60: Attributes Inherited by the Toggle Button Gadget

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

Table A-60: (continued)

Attribute Name	Data Type	Default
Label Attributes		
DwtNlabel	DwtCompString	Widget name
DwtNalignment	unsigned char	DwtAlignmentCenter
DwtNdirectionRToL	Boolean	False
DwtNhelpCallback	DwtCallbackPtr	NULL

Table A-61: Widget-Specific Attributes for the the Toggle Button Gadget

Attribute Name	Data Type	Default
DwtNshape	unsigned char	DwtRectangular
DwtNvalue	Boolean	False
DwtNvisibleWhenOff	Boolean	True
DwtNvalueChangedCallback	DwtCallbackPtr	NULL
DwtNbuttonAccelerator	char *	NULL
DwtNacceleratorText	DwtCompString	NULL

A.31 Window Widget

Table A-62: Attributes Inherited by the Window Widget

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

Table A-62: (continued)

Attribute Name	Data Type	Default
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	

Table A-63: Widget-Specific Attributes for the Window Widget

Attribute Name	Data Type	Default
DwtNexposeCallback	DwtCallbackPtr	NULL

A.32 Work-in-Progress Box Widget

Table A-64: Attributes Inherited by the Work-in-Progress Box Widget

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

Table A-64: (continued)

Attribute Name	Data Type	Default
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	

Table A-65: Widget-Specific Attributes for the Work-in-Progress Box Widget

Attribute Name	Data Type	Default
DwtNlabel	DwtCompString	Widget name
DwtNcancelLabel	DwtCompString	"Cancel"
DwtNcancelCallback	DwtCallbackPtr	NULL

Index

C

D

E

F

G

H

I

K

L

M

P

R

S

T

W

X

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[1]	————	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
Guide to the XUI Toolkit:
C Language Binding
AA-MA95B-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
Guide to the XUI Toolkit:
C Language Binding
AA-MA95B-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