Widget Reference (T – Z) 10

TextEdit Widget

Class

Class Name: `TextEdit`
Class Pointer: `textEditWidgetClass`

Ancestry

Core-Primitive-TextEdit

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/TextEdit.h>
```

Description

The TextEdit widget provides a multi-line text editing facility that has both a customizable user interface and a programmatic interface. It provides a consistent editing paradigm for textual data. An example is shown in the following figure.

10

TextEdit Widget

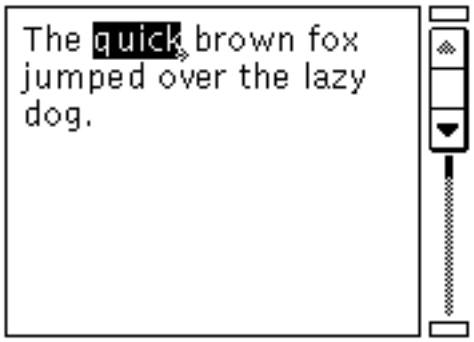

Figure 10-1 Simple TextEdit Widget

The TextEdit widget provides three text wrap modes: OL_WRAP_OFF, OL_WRAP_ANY, and OL_WRAP_WHITE_SPACE. The TextEdit widget manages its text data using a `TextBuffer` data structure and allows the application to access and manipulate its TextBuffer in order to implement more complex textual operations.

The TextEdit widget provides several distinct callback lists used to monitor the state of the textual data: insertion cursor movement, modification of the text, and post modification notification. Each of these callbacks provide information to the application regarding the intended action. The application can simply examine this information to maintain its current state or can disallow the action and perform any of the programmatic manipulations instead.

The TextEdit widget provides distinct callback lists for user input: mouse button down and key press. The *call_data* for these callbacks decodes the input for the application. The application can examine the input and either consume the action, and perform any of the programmatic manipulations, or allow the widget to act upon it.

The TextEdit widget also provides the application with a callback list invoked when the widget is redisplayed. With this callback the application can add callbacks that can be used to display information in the margins of the TextEdit, such as line numbers or update marks. (See “XtNmargin” on page 636.)

Editing Capabilities

The TextEdit widget provides editing capabilities to move the insert point, select text, delete text, scroll the display, perform cut, copy, paste, and undo operations, and refresh the text display. All of these capabilities are represented by OLIT Activation Types, which are mapped to key bindings via global resources stored in the X server. All of these settings dynamically change immediately after new resource values are stored in the server. See Table 10-4 on page 642 for a complete list of these Activation Types supported by the TextEdit widget.

Text Hierarchy

Text is considered to be hierarchically composed of white space, words, lines, and paragraphs. These terms are defined as:

white space	Any non-empty sequence of the ASCII characters space, tab, linefeed, or carriage return (decimal values of 32, 9, 10, 13, respectively)
word	Any non-empty sequence of characters bounded on both sides by white space.
source line	Any (possibly empty) sequence of characters bounded by newline characters.
display line	Any (possibly empty) sequence of characters appearing on a single window display line.
source paragraph	Any sequence of characters bounded by sets of two or more adjacent newline characters.
display paragraph	Any (possibly empty) sequence of characters bounded by newline characters. (This is a synonym for *source line*.)

In all cases, the beginning or end of the edit text is an acceptable bounding element in the previous definitions.

The Text Buffer

The TextEdit widget uses a *TextBuffer* to store and manage the text data. If the application needs to implement complex operations on the text (i.e., search for a string, replace a paragraph, etc.), it can access the handle to the TextEdit's TextBuffer and then use the set of TextBuffer routines to manipulate the TextBuffer. Since the TextBuffer provides more detailed information on the

structure of the text (words, lines, pages, etc.) than the TextEdit widget itself, the application can use the TextBuffer to have more programmatic control over the text. There are two variants of the TextBuffer that are used, depending on the value of the widget's `XtNtextFormat` resource. See "Text Buffer Functions" on page 163 and "Text Buffer Functions for Internationalization" on page 176 for information on the TextBuffer.

Sizing the Display

When making display decisions, the TextEdit widget first will use either the application-specified width and height or, if these values are not specified, calculate width and height by applying the values of the `XtNcharsVisible` and `XtNlinesVisible` resources. Once the width and height are determined, the TextEdit widget will request an appropriate size from its parent (considering the margins). If the request is denied or only partially satisfied, no future growth requests will be made unless there is an intervening resize operation externally imposed.

Once the size of the display is settled, the TextEdit widget calculates the display lines based on this size, the various margins, the font, tab table, and wrap mode.

Wrapping

If the wrap mode (see "XtNwrapMode" on page 642) is OL_WRAP_ANY, as many characters from the source line as will entirely fit before the right margin are written to the current display line, then the next character starts at the left margin of the next display line, and so on.

If the wrap mode is OL_WRAP_WHITE_SPACE, the line wrap occurs at the first whitespace character that follows the last full word that does fit on the current display line. If the first full word that does not fit is the first word on the display line, however, the wrap is made as if OL_WRAP_ANY were selected.

If the wrap mode is OL_WRAP_OFF, the lines are not wrapped but are clipped at the right margin. In this mode the text is horizontally scrollable.

Text Scrolling

The application can get horizontal and/or vertical scrolling of a TextEdit by creating it as a child of a ScrolledWindow widget; see "ScrolledWindow Widget" on page 529. The ScrolledWindow will completely manage the scrolling of text for the application by using its scrollbars.

The proportion indicators on the scrollbars show relatively how much of the text is currently in the display.

As the user enters text, the view automatically scrolls when the insert point moves beyond a margin boundary (right or bottom) to keep the insert point in view.

Coloration

The following diagram illustrates the resources that affect TextEdit coloration.

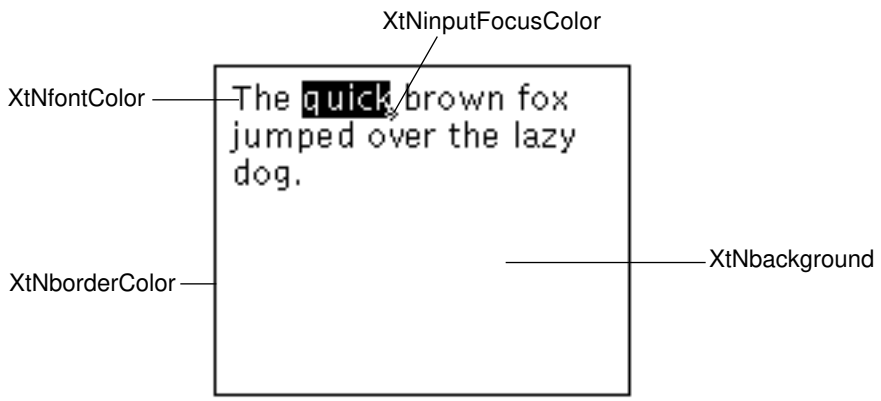

Figure 10-2 TextEdit Coloration

Keyboard Traversal

The default value of the `XtNtraversalOn` resource is TRUE.

The TextEdit widget responds to the following Activation Types for keyboard navigation. For more information on these Activation Types and their key bindings, see Table 10-4 on page 642.

OL_CHARBAK	OL_LINESTART	OL_ROWDOWN
OL_CHARFWD	OL_NEXTFIELD	OL_ROWUP
OL_DOCEND	OL_PANEEND	OL_WORDBAK
OL_DOCSTART	OL_PANESTART	OL_WORDFWD
OL_LINEEND	OL_PREVFIELD	

It is expected that the user will use the alternate bindings for NEXTFIELD and PREVFIELD because the primary binding, <Tab> and <Shift-Tab>, are valid characters in a Text pane (unless `XtNinsertTabs` is set to FALSE).

The TextEdit widget responds to the following Activation Types for selection. For more information on these Activation Types and their key bindings, see Table 10-4 on page 642.

OL_SELCHARBAK	OL_SELLINE	OL_SELWORDBAK
OL_SELCHARFWD	OL_SELLINEBAK	OL_SELWORDFWD
OL_SELFLIPENDS	OL_SELLINEFWD	

The TextEdit widget responds to the following Activation Types for scrolling. For more information on these Activation Types and their key bindings, see Table 10-4 on page 642.

OL_PAGEDOWN	OL_SCROLLBOTTOM	OL_SCROLLRIGHT
OL_PAGELEFT	OL_SCROLLDOWN	OL_SCROLLRIGHTEDGE
OL_PAGERIGHT	OL_SCROLLLEFT	OL_SCROLLTOP
OL_PAGEUP	OL_SCROLLLEFTEDGE	OL_SCROLLUP

The TextEdit widget responds to the following Activation Types for editing. For more information on these Activation Types and their key bindings, see Table 10-4 on page 642.

OL_DELCHARBAK	OL_DELLINEBAK	OL_DELWORDFWD
OL_DELCHARFWD	OL_DELLINEFWD	OL_UNDO
OL_DELLINE	OL_DELWORDBAK	

Keyboard Mnemonic Display

The TextEdit does not display the mnemonic. If the TextEdit widget is the child of a Caption widget, the Caption widget can be used to display the mnemonic.

Keyboard Accelerator Display

The TextEdit does not display the keyboard accelerator. If the TextEdit is the child of a Caption widget, the Caption widget can be used to display the accelerator as part of the label.

Resources

Table 10-1 TextEdit Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	Int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 10-2 TextEdit Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	SGI
XtNacceleratorText	String	NULL	n/a
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGID
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	(calculated; see page 27)	SGID
XtNmnemonic	Unsigned char	'\0'	n/a
XtNreferenceName	String	NULL	GI

Table 10-2 TextEdit Primitive Resources (Continued)

Name	Type	Default	Access
XtNreferenceWidget	Widget	NULL	GI
XtNscale	Int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 10-3 TextEdit Resources

Name	Type	Default	Access
XtNblinkRate	long	666	SGI
XtNbottomMargin	Dimension	4	SGI
XtNbuttons	XtCallbackList	NULL	SGIO
XtNcharsVisible	Int	50	GI
XtNcopyLabel	OlStr	“Copy”	GI
XtNcopyMnemonic	unsigned char	‘\0’	GI
XtNcursorPosition	TextPosition	0	SGI
XtNcutLabel	OlStr	“Cut”	GI
XtNcutMnemonic	Unsigned char	‘\0’	GI
XtNdeleteLabel	OlStr	“Delete”	GI
XtNdeleteMnemonic	unsigned char	‘\0’	GI
XtNdisplayPosition	TextPosition	0	SGI
XtNeditType	OlEditMode	OL_TEXT_EDIT	SGI
XtNgrowMode	OlDefine	OL_GROW_OFF	SGI
XtNimPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI
XtNinsertTab	Boolean	TRUE	SGI
XtNkeys	XtCallbackList	NULL	SGIO
XtNleftMargin	Dimension	4	SGI
XtNlinesVisible	Int	16	GI
XtNmargin	XtCallbackList	NULL	SGIO
XtNmenuTitle	OlStr	“Edit”	GI
XtNmodifyVerification	XtCallbackList	NULL	SGIO
XtNmotionVerification	XtCallbackList	NULL	SGIO
XtNpasteLabel	OlStr	“Paste”	GI
XtNpasteMnemonic	Unsigned char	‘\0’	GI

Table 10-3 TextEdit Resources (Continued)

Name	Type	Default	Access
XtNpostModifyNotification	XtCallbackList	NULL	SGIO
XtNrightMargin	Dimension	4	SGI
XtNselectEnd	TextPosition	0	SGI
XtNselectStart	TextPosition	0	SGI
XtNsource	OlStr	NULL	SGI
XtNsourceType	OlSourceType	OL_STRING_SOURCE	SGI
XtNtabTable	TabTable	NULL	SGI
XtNtopMargin	Dimension	4	SGI
XtNundoLabel	OlStr	"Undo"	GI
XtNundoMnemonic	Unsigned char	'\0'	GI
XtNwrapMode	OlWrapMode	OL_WRAP_WHITE_SPACE	SGI

XtNblinkRate

Class	Type	Default	Access
XtCBlinkRate	Long	666	SGI

Synopsis: The rate that the active input caret blinks.
Values: The number of milliseconds between blinks. Setting this value to zero turns off the blink effect.

XtNbottomMargin

Class	Type	Default	Access
XtCMargin	Dimension	4	SGI

Synopsis: The number of pixels used for the height of the bottom margin.

XtNbuttons

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when a mouse button press is made in the pane.

The *call_data* parameter is a pointer to an `OlInputCallData` structure:

```
typedef struct {
    Boolean         consumed;
    XEvent          *event;
    KeySym          *keysym;
    char            **buffer;
    int             *length;
    OlInputEvent    ol_event;
} OlInputCallData, *OlInputCallDataPointer;

typedef enum {
    OL_UNKNOWN_INPUT, OL_SELECT, OL_ADJUST, OL_MENU,
    OL_CONSTRAIN, OL_DUPLICATE, OL_PAN, OL_UNKNOWN_KEY_INPUT,
    OL_CUT, OL_COPY, OL_PASTE, OL_HELP, OL_CANCEL, OL_PROP,
    OL_STOP, OL_UNDO, OL_NEXT_FIELD, OL_PREV_FIELD, OL_CHARFWD,
    OL_CHARBAK, OL_ROWDOWN, OL_ROWUP, OL_WORDFWD, OL_WORDBAK,
    OL_LINESTART, OL_LINEEND, OL_DOCSTART, OL_DOCEND, OL_PANESTART,
    OL_PANEEND, OL_DELCHARFWD, OL_DELCHARBAK, OL_DELWORDFWD,
    OL_DELWORDBAK, OL_DELLINEFWD, OL_DELLINEBAK, OL_DELLINE,
    OL_SELCHARFWD, OL_SELCHARBAK, OL_SELWORDFWD, OL_SELWORDBAK,
    OL_SELLINEFWD, OL_SELLINEBAK, OL_SELLINE, OL_SELFLIPENDS,
    OL_REDRAW, OL_RETURN, OL_PAGEUP, OL_PAGEDOWN, OL_HOME, OL_END,
    OL_SCROLLUP, OL_SCROLLDOWN, OL_SCROLLLEFT, OL_SCROLLRIGHT,
    OL_SCROLLLEFTEDGE, OL_SCROLLRIGHTEDGE, OL_PGM_GOTO
} OlInputEvent;
```

This callback can be used to remap or consume button events; for example, to override the default Edit menu. To do so, the application would add a `XtNbuttons` callback that checks the *ol_event* member of the `OlInputCallData` structure. If the value of the *ol_event* member is OL_MENU, the callback posts the application-defined menu and sets the *consumed* member of the `OlInputCallData` structure to TRUE.

XtNcharsVisible

Class	Type	Default	Access
XtCCharsVisible	int	50	GI

Synopsis: The initial width of the text in terms of characters.
Values: 0 ≤ `XtNcharsVisible`

This resource overrides the `XtNwidth` resource setting. The `XtNwidth` is recalculated to be the value of `XtNcharsVisible` multiplied by the width of the "n" (en) character in the font plus the values for the left and right margins.

The value of this resource changes to reflect the effects of geometry changes imposed by the widget tree and the user. Calls to `XtSetValues()` for this resource are ignored.

XtNcopyLabel/
XtNcutLabel/
XtNdeleteLabel/
XtNpasteLabel/
XtNundoLabel

Class	Type	Default	Access
XtCCopyLabel	OlStr	"Copy"	GI
XtCCutLabel	OlStr	"Cut"	GI
XtCDeleteLabel	OlStr	"Delete"	GI
XtCPasteLabel	OlStr	"Paste"	GI
XtCUndoLabel	OlStr	"Undo"	GI

Synopsis: The label for the Copy, Cut, Delete, Paste, and Undo buttons in the Edit menu, respectively.
Values: Any `OlStr` value valid in the current locale.

XtNcopyMnemonic/
XtNcutMnemonic/
XtNdeleteMnemonic/
XtNpasteMnemonic/
XtNundoMnemonic

Class	Type	Default	Access
XtCCopyMnemonic	unsigned char	'\0'	GI
XtCCutMnemonic	unsigned char	'\0'	GI
XtCDeleteMnemonic	unsigned char	'\0'	GI
XtCPasteMnemonic	unsigned char	'\0'	GI
XtCUndoMnemonic	unsigned char	'\0'	GI

Synopsis: The mnemonic for the Copy, Cut, Delete, Paste, and Undo buttons in the Edit menu, respectively.
Values: Any ASCII character.

XtNcursorPosition

Class	Type	Default	Access
XtCTextPosition	TextPosition	0	SGI

Synopsis: The relative character position in the text of the insert cursor.
Values: 0 ≤ `XtNcursorPosition` < *number-of-chars-in-buffer*

Changing the value of this resource may affect the `XtNdisplayPosition` resource if the `XtNcursorPosition` value is not visible in the pane.

XtNdisplayPosition

Class	Type	Default	Access
XtCTextPosition	TextPosition	0	SGI

Synopsis: The character position in the text that will be displayed at the top of the pane.
Values: 0 ≤ `XtNdisplayPosition` < *number-of-chars-in-textbuffer*

A value of 0 indicates the beginning of the text source. When the value provided is near the end of the buffer, this position is recalculated to ensure that the last line in the buffer appears as the last line in the pane.

XtNeditType

Class	Type	Default	Access
XtCEditType	OlEditMode	OL_TEXT_EDIT	SGI

Synopsis: The edit state of the widget.
Values: `OL_TEXT_READ/"textread"` – The contents are read-only; the user cannot edit any text.
`OL_TEXT_EDIT/"textedit"` – The text is fully editable.

XtNgrowMode

Class	Type	Default	Access
XtCGrowMode	OlDefine	OL_GROW_OFF	SGI

Synopsis: The resize policy of the widget (in conjunction with `XtNwrapmode`).
Values: `OL_GROW_OFF/"grow_off"` – The widget will not grow either horizontally or vertically when text exceeds window boundaries.
`OL_GROW_HORIZONTAL/"grow_horizontal"` – The widget will grow horizontally, if the text exceeds the width of the window. This setting is valid only when `XtNwrapMode` is OL_WRAP_OFF.

OL_GROW_VERTICAL/"grow_vertical" – The widget will grow vertically, if the text exceeds the height of the window.
OL_GROW_BOTH/"grow_both" – The widget will grow both horizontally and vertically as required, if the text exceeds the window width or height. This setting is valid only when XtNwrapMode is OL_WRAP_OFF.

XtNimPreeditStyle

Class	Type	Default	Access
XtCImPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI

Synopsis: The pre-edit style (in conjunction with the shell's XtNimStatusStyle resource). If the pre-edit style is not supported by the input method, the ability to pre-edit is lost.

Values: OL_ON_THE_SPOT/"onTheSpot" – The pre-edit data is displayed at the insertion point in the application window. The preexisting user data is shifted and the pre-edit data is inserted at the point of insertion.
OL_OVER_THE_SPOT/"overTheSpot" – The pre-edit data is displayed in the application window, starting at the insertion point. As the user types the pre-edit data, the preexisting user data is obscured by the pre-edit data.
OL_ROOT_WINDOW/"rootWindow" – The pre-edit data is displayed in a child of the root window, away from the point of insertion.
OL_NO_PREEDIT/"none" – No pre-edit data is displayed.

See "XtNimStatusStyle" on page 44 and "Setting the Input Method Pre-Edit and Status Styles (Asian Locales Only)" on page 82.

XtNinsertTab

Class	Type	Default	Access
XtCInsertTab	Boolean	TRUE	SGI

Synopsis: Determines whether a tab character is insertable.

Values: TRUE/"true" – Tabs are insertable but not used for forward traversal. Forward traversal can still be accomplished with Control-Tab.
FALSE/"false"– Tabs are used for forward traversal and are not insertable.

XtNkeys

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when a key press is made in the pane.

The *call_data* parameter is a pointer to an `OlInputCallData` structure, as shown in "XtNbuttons" on page 631.

This callback can be used to remap or consume key events, such as to post a property window for the text. To do so, the application would add a `XtNkeys` callback that checks the *ol_event* member of the `OlInputCallData` structure. If the value of the *ol_event* member is OL_PROP, the callback posts the property window (or raises it if it is already mapped) and sets the *consumed* member of the `OlInputCallData` structure to TRUE.

XtNleftMargin

Class	Type	Default	Access
XtCMargin	Dimension	4	SGI

Synopsis: The number of pixels used for the width of the left margin.

XtNlinesVisible

Class	Type	Default	Access
XtCLinesVisible	int	16	GI

Synopsis: The initial height of the text in terms of display lines.
Values: 1 ≤ `XtNlinesVisible`

This resource overrides the `XtNheight` resource setting. The `XtNheight` is recalculated to be the value of `XtNlinesVisible` multiplied by the height of the font plus the values for the top and bottom margins.

The value of `XtNlinesVisible` changes to reflect the effects of geometry changes imposed by the widget tree and the user. Calls to `XtSetValues()` for this resource are ignored.

XtNmargin

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the pane is redisplayed.

The *call_data* parameter is a pointer to an `OlTextMarginCallData` structure:

```
typedef struct {
    OlTextMarginHint   hint;
    XRectangle         *rect;
} OlTextMarginCallData, *OlTextMarginCallDataPointer;

typedef enum {
    OL_MARGIN_EXPOSED,
    OL_MARGIN_CALCULATED
} OlTextMarginHint;
```

The *hint* member indicates whether the area to be redrawn was explicitly known because of an exposure event (OL_MARGIN_EXPOSED) or if the rectangle was calculated relative to the textual display OL_MARGIN_CALCULATED). The margin callback should respond to the OL_MARGIN_EXPOSED hint by repainting the area defined by the *rect* member.

The margin callback may wish to calculate its own rectangle in the OL_MARGIN_CALCULATED case. It can use the rectangle structure passed with the *call_data* for this purpose. This callback can be used to repair the margins for the text, such as to display line numbers for the text in the left margin.

XtNmenuTitle

Class	Type	Default	Access
XtCMenuTitle	OlStr	"Edit"	GI

Synopsis: The title of the TextEdit menu.
Values: Any `OlStr` value valid in the current locale.

XtNmodifyVerification

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when a modification of the text is attempted.

The *call_data* parameter is a pointer to an `OlTextModifyCallData` structure:

```
typedef struct {
    Boolean         ok;
    TextPosition    current_cursor;
    TextPosition    select_start;
    TextPosition    select_end;
    TextPosition    new_cursor;
    TextPosition    new_select_start;
```

```
    TextPosition    new_select_end;
    char            *text;
    int             text_length;
} OlTextModifyCallData, *OlTextModifyCallDataPointer;
```

All of the fields in this structure, with the exception of the *ok* flag, are treated as read-only information. The application can return without changing the value of *ok* (initially TRUE), in which case the update will occur. The application can also set the *ok* flag to FALSE, perform any other operations it desires, and return, in which case the update will not be performed.

XtNmotionVerification

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked whenever the cursor position moves within the widget.

The *call_data* parameter is a pointer to an `OlTextMotionCallData` structure:

```
typedef struct {
    Boolean         ok;
    TextPosition    current_cursor;
    TextPosition    new_cursor;
    TextPosition    select_start;
    TextPosition    select_end;
} OlTextMotionCallData, *OlTextMotionCallDataPointer;
```

This callback list is used whenever the cursor position changes due to cursor movement operations or by modification of the text.

The application can distinguish between a simple cursor movement and a modify operation by comparing the *current_cursor* and *new_cursor* values. When these values are equal, the callback is the result of a modify operation. In this case the *ok* flag is ignored and the application should not attempt to perform updates to the text or its display during this callback.

If the values of *current_cursor* and *new_cursor* are different, the application is guaranteed that the operation is the result of a cursor movement. In this mode all of the fields in this structure, with the exception of the *ok* flag, are treated as read-only information.

The application can return without changing the value of the *ok* flag (initially TRUE), in which case the movement will occur. The application can also set *ok* to FALSE, perform any other operations it desires and return, in which case the movement will not be performed.

XtNpostModifyNotification

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked after a text update has completed.

The *call_data* parameter is a pointer to an `OlTextPostModifyCallData` structure:

```
typedef struct {
    Boolean         requestor;
    TextPosition    new_cursor;
    TextPosition    new_select_start;
    TextPosition    new_select_end;
    char            *inserted;
    char            *deleted;
    TextLocation    delete_start;
    TextLocation    delete_end;
    TextLocation    insert_start;
    TextLocation    insert_end;
    TextLocation    cursor_position;
} OlTextPostModifyCallData, *OlTextPostModifyCallDataPointer;
```

This callback synchronizes the application with the text once a modify operation is completed. For example, the application may record successful edit operations in an undo buffer to provide multilevel undo functionality. The data provided in this callback is considered read-only and volatile (the application should copy what it needs from this structure before returning).

XtNrightMargin

Class	Type	Default	Access
XtCMargin	Dimension	4	SGI

Synopsis: The size of the right margin, in pixels.

XtNselectEnd / XtNselectStart

Class	Type	Default	Access
XtCTextPosition	TextPosition	0	SGI
XtCTextPosition	TextPosition	0	SGI

Synopsis: The character position of the end/start of the current text selection.

This resource is used with `XtNselectStart` and `XtNcursorPosition` to specify a selection. To be effective, the following conditions must be true:

```
XtNselectStart ≤ XtNselectEnd
      and either XtNcursorPosition == XtNselectStart
                 or
                 XtNcursorPosition == XtNselectEnd
```

If either of these tests fails, then `XtNselectStart` and `XtNselectEnd` are set to the value of `XtNcursorPosition`.

XtNsource

Class	Type	Default	Access
XtCSource	OlStr	NULL	SGI

Synopsis: The source of the text to display.
Values: The datatype of the value depends on the value of `XtNsourceType`:

XtNsourceType	XtNsource Contents
OL_STRING_SOURCE	Any `OlStr` value valid in the current locale.
OL_DISK_SOURCE	A string that represents the pathname of a file containing the source text.
OL_TEXT_BUFFER_SOURCE	A pointer to a TextBuffer containing the source text in single-byte format.
OL_OLTEXT_BUFFER_SOURCE	A pointer to a TextBuffer containing the source text in multi-byte format.

The TextEdit widget does not guarantee that an `XtGetValues()` on this resource will return the correct data and datatype. Internally, the passed-in values are overwritten. To get at the current text contents, use the TextEdit function `OlTextEditCopyBuffer()` (see page 660).

XtNsourceType

Class	Type	Default	Access
XtCSourceType	OlSourceType	OL_STRING_SOURCE	SGI

Synopsis: The interpretation of the `XtNsource` resource value.

Values: `OL_STRING_SOURCE/"stringsource"` – The `XtNsource` value is interpreted as the string to be used as the source.
`OL_DISK_SOURCE/"disksource"` – The `XtNsource` value is interpreted as the name of the file containing the source.
`OL_TEXT_BUFFER_SOURCE` – The `XtNsource` value is interpreted as a pointer to a previously initialized `single-byte format TextBuffer` (see "Text Buffer Functions" on page 163 for a description of TextBuffers). This value can only be set programmatically; there is no corresponding resource file string.
`OL_OLTEXT_BUFFER_SOURCE` – The `XtNsource` value is interpreted as a pointer to a previously initialized `multi-byte format TextBuffer` (see "Text Buffer Functions" on page 163 for a description of TextBuffers). This value can only be set programmatically; there is no corresponding resource file string.

XtNtabTable

Class	Type	Default	Access
XtCTabTable	TabTable	NULL	SGI

Synopsis: The pointer to an array of tab positions.

The tab positions are specified in terms of pixels and must be terminated by a zero (0) entry. The widget calculates tabs by finding the next tab table entry that exceeds the current x offset for the line. If no such entry exists in the table or if the pointer to the tab table is NULL, the tab is set to the next greater multiple of 8 times the size of the "n" (en) character in the font.

XtNtopMargin

Class	Type	Default	Access
XtCMargin	Dimension	4	SGI

Synopsis: The number of pixels used for the top margin.

XtNwrapMode

Class	Type	Default	Access
XtCWrapMode	OlWrapMode	OL_WRAP_WHITE_SPACE	SGI

Synopsis: The wrapping of the text in the pane.
Values: `OL_WRAP_ANY/"wrapany"` – Lines are wrapped at the last character before the right margin.
`OL_WRAP_WHITE_SPACE/"wrapwhitespace"` – Lines are wrapped at the last white space before the right margin or at the last character before the right margin if the line does not contain any white space.
`OL_WRAP_OFF/"wrapoff"` – Lines are not wrapped and the pane may scroll horizontally.

Activation Types

The following table lists the activation types used by the TextEdit.

Table 10-4 TextEdit Activation Types

Activation Type	Semantics	Resource Name
OL_ADJUST	ADJUST	XtNadjustBtn
OL_ADJUSTKEY	ADJUST	XtNadjustKey
OL_CANCEL	CANCEL	XtNcancelKey
OL_CHARBAK	LEFT	XtNleftKey
OL_CHARFWD	RIGHT	XtNrightKey
OL_COPY	COPY	XtNcopyBtn
OL_CUT	CUT	XtNcutBtn
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_DELCHARBAK	DELETEBACKWARD	XtNdelCharBakFwd
OL_DELCHARFWD	DELETEFORWARD	XtNdelCharFwdKey
OL_DELLINE	DELETELINE	XtNdelLineKey
OL_DELLINEBAK	DELLINEBAK	XtNdelLineBakKey
OL_DELLINEFWD	DELLINEFWD	XtNdelLineFwdKey
OL_DELWORDBAK	DELWORDBAK	XtNdelWordBakKey
OL_DELWORDFWD	DELWORDFWD	XtNdelWordFwdKey
OL_DOCEND	DATAEND	XtNdocEndKey
OL_DOCSTART	DATASTART	XtNdocStartKey

Table 10-4 TextEdit Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_HELP	HELP	XtNhelpKey
OL_LINEEND	ROWEND	XtNlineEndKey
OL_LINESTART	ROWSTART	XtNlineStartKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PAGEDOWN	PAGEDOWN	XtNpageDownKey
OL_PAGELEFT	PAGELEFT	XtNpageLeftKey
OL_PAGERIGHT	PAGERIGHT	XtNpageRightKey
OL_PAGEUP	PAGEUP	XtNpageUpKey
OL_PANEEND	PANEDOWN	XtNpaneEndKey
OL_PANESTART	PANEUP	XtNpaneStartKey
OL_PASTE	PASTE	XtNpasteBtn
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_ROWDOWN	DOWN	XtNdownKey
OL_ROWUP	UP	XtNupKey
OL_SCROLLDOWN	SCROLLDOWN	XtNscrollDownKey
OL_SCROLLLEFT	SCROLLLEFT	XtNscrollLeftKey
OL_SCROLLLEFTEDGE	SCROLLLEFTEDGE	XtNscrollLeftEdgeKey
OL_SCROLLRIGHT	SCROLLRIGHT	XtNscrollRightKey
OL_SCROLLRIGHTEDGE	SCROLLRIGHTEDGE	XtNscrollRightEdgeKey
OL_SCROLLUP	SCROLLUP	XtNscrollUpKey
OL_SELCHARBAK	SELCHARBAK	XtNselCharBakKey
OL_SELCHARFWD	SELCHARFWD	XtNselCharFwdKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_SELFLIPENDS	SELFLIPENDS	XtNselFlipEndsKey
OL_SELLINE	SELLINE	XtNselLineKey
OL_SELLINEBAK	SELLINEBAK	XtNselLineBakKey

Table 10-4 TextEdit Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_SELLINEFWD	SELLINEFWD	XtNselLineFwdKey
OL_SELWORDBAK	SELWORDBAK	XtNselWordBakKey
OL_SELWORDFWD	SELWORDFWD	XtNselWordFwdKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey
OL_UNDO	UNDO	XtNundoKey
OL_WORDBAK	JUMPLEFT	XtNwordBakKey
OL_WORDFWD	JUMPRIGHT	XtNwordFwdKey

Activation types not described in the following list are described in "Common Activation Types" on page 68.

OL_ADJUST

The OL_ADJUST activation type first calls the `XtNbuttons` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNbuttons` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The OL_ADJUST activation type calls the `XtNmotionVerification` callback list with the appropriate `OlTextMotionCallData` structure containing the new cursor position, selection start, and selection end. If a callback is not registered or the `OlTextMotionCallData`'s *ok* field is TRUE, the OL_ADJUST activation type will modify the `XtNcursorPosition`, `XtNselectStart`, and `XtNselectEnd` resources to reflect the cursor position and the resulting selection.

OL_CHARBAK

The OL_CHARBAK activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved one character before the current cursor position. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved. When the cursor is positioned before the first character on a line, the

OL_CHARBAK activation type will cause the previous line to be brought into view and the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_CHARFWD

The OL_CHARFWD activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved one character after the current cursor position. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved. When the cursor is positioned after the last character on a line, the activation type will cause the next line to be brought into view and the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_COPY

The OL_COPY activation type first calls the XtNbuttons callback list with the appropriate OlInputCallData structure. If there is no XtNbuttons callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmodifyVerification callback list with an OlTextModifyCallData structure containing the currently selected items. If the OlTextModifyCallData *ok* field is TRUE upon return, then the selected text will be copied to the CLIPBOARD.

OL_CUT

The OL_CUT activation type first calls the XtNbuttons callback list with the appropriate OlInputCallData structure. If there is no XtNbuttons callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type applies only to TextEdit widgets that have XtNeditType of OL_TEXT_EDIT. The XtNmodifyVerification callback list will be called with an OlTextModifyCallData structure containing the currently selected items. If the OlTextModifyCallData *ok* field is TRUE upon return, then the selected text will be copied to the CLIPBOARD.

OL_DELCHARBAK

The OL_DELCHARBAK activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the character to the left of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the character has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELCHARFWD

The OL_DELCHARFWD activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the character to the right of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the character has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELLINE

The OL_DELLINE activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the current line has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the line has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELLINEBAK

The OL_DELLINEBAK activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the line to the left of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the line has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELLINEFWD

The OL_DELLINEFWD activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the line to the right of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the line has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELWORDBAK

The OL_DELWORDBAK activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the word to the left of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the word has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELWORDFWD

The OL_DELWORDFWD activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the word to the right of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the word has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DOCEND

The OL_DOCEND activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved to after the last character on the last line of the TextBuffer. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved and the text may be scrolled to bring the last line into view. If the text is scrolled to bring the last line into view, the `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_DOCSTART

The OL_DOCSTART activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved to before the first character on the first line of the TextBuffer.

If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved and the text may be scrolled to bring the first line into view. If the text is scrolled to bring the first line into view, the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_LINEEND

The OL_LINEEND activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved to after the last character on the current line. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved.

OL_LINESTART

The OL_LINESTART activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved to before the first character on the current line. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved.

OL_MENU/ OL_MENUKEY

The OL_MENU activation type first calls the XtNbuttons callback list with the appropriate OlInputCallData structure. If there is no XtNbuttons callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The OL_MENUKEY activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The OL_MENU and OL_MENUKEY activation types will pop up the TextEdit menu.

OL_PAGEDOWN

The OL_PAGEDOWN activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the page after the current view into the view.

OL_PAGELEFT

The OL_PAGELEFT activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The OL_PAGELEFT activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the page to the left of the current view into the view.

OL_PAGERIGHT

The OL_PAGERIGHT activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The OL_PAGERIGHT activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the page to the right of the current view into the view.

OL_PAGEUP

The OL_PAGEUP activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The OL_PAGEUP activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the page before the current view into the view.

OL_PANEEND

The OL_PANEEND activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor

position moved to after the last character on the last line in the current view. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved.

OL_PANESTART

The OL_PANESTART activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved to before the first character on the first line in the current view. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved.

OL_PASTE

The OL_PASTE activation type first calls the XtNbuttons callback list with the appropriate OlInputCallData structure. If there is no XtNbuttons callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The OL_PASTE activation type applies only to TextEdit widgets that have XtNeditType of OL_TEXT_EDIT. When the widget has focus, then the OL_PASTE activation type will insert the contents of the CLIPBOARD at the current insert position by calling the XtNpostModifyVerification callback list. The XtNmotionVerification callback list will be called with an OlTextMotionCallData structure that represents the new cursor and selection position.

OL_ROWDOWN

The OL_ROWDOWN activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The OL_ROWDOWN activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved one line after the current cursor position. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved and the text may be scrolled to bring the new line into view. When a new line is brought into the view, the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_ROWUP

The OL_ROWUP activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved one line previous to the current cursor position. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved and the text may be scrolled to bring the new line into view. When a new line is brought into the view, the `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_SCROLLDOWN

The OL_SCROLLDOWN activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the line after the current view into the view.

OL_SCROLLLEFT

The OL_SCROLLLEFT activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the character left of the current view into the view.

OL_SCROLLLEFTEDGE

The OL_SCROLLLEFTEDGE activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the left-most character of the document into the view.

OL_SCROLLRIGHT

The OL_SCROLLRIGHT activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the character right of the current view into the view.

OL_SCROLLRIGHTEDGE

The OL_SCROLLRIGHTEDGE activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the rightmost character of the document into the view.

OL_SCROLLUP

The OL_SCROLLUP activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the line before the current view into the view.

OL_SELCHARBAK

The OL_SELCHARBAK activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended one character to the left of the cursor position and the cursor position moved to before the selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the selection will be extended, the cursor will be moved, and the text may be scrolled to bring the previous line into view. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_SELCHARFWD

The OL_SELCHARFWD activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended one character to the right of the cursor position and the cursor position moved to after the selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the selection will be extended, the cursor will be moved, and the text may be scrolled to bring the next line into view. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_SELECT

The OL_SELECT activation type first calls the `XtNbuttons` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNbuttons` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with the appropriate `OlTextMotionCallData` structure containing the new cursor position, selection start, and selection end. If a callback is not registered or the `OlTextMotionCallData`'s *ok* field is TRUE, the OL_SELECT activation type will modify the `XtNcursorPosition`, `XtNselectStart`, and `XtNselectEnd` resources to reflect the cursor position and the resulting selection.

OL_SELFLIPENDS

The OL_SELFLIPENDS activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved to the opposite end of the current selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved. If the text is scrolled to bring the cursor into view, the `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_SELLINE

The OL_SELLINE activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended to the beginning of the current line and to the end of the current line. and the cursor position moved to before the selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the selection will be extended. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_SELLINEBAK

The OL_SELLINEBAK activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended to the beginning of the line to the left of the current cursor position and the cursor position moved to before the selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the selection will be extended, the cursor will be moved, and the text may be scrolled to bring the previous line into view. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_SELLINEFWD

The OL_SELLINEFWD activation type first calls the `XtNkeys` callback list with the appropriate `OlInputCallData` structure. If there is no `XtNkeys` callback or if the *consumed* field of the `OlInputCallData` structure is TRUE, then processing continues. The activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended to the end of the line to the right of the current cursor position and the cursor position moved to after the selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the selection will be

extended, the cursor will be moved, and the text may be scrolled to bring the next line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELWORDBAK

The OL_SELWORDBAK activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended to the beginning of the word to the left of the current cursor position and the cursor position moved to before the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, the cursor will be moved, and the text may be scrolled to bring the previous line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELWORDFWD

The OL_SELWORDFWD activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended to the end of the word to the right of the current cursor position and the cursor position moved to after the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, the cursor will be moved, and the text may be scrolled to bring the next line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_UNDO

The OL_UNDO activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmodifyVerification callback list with an OlTextModifyCallData structure that represents the selection, cursor position, and text before the last modification. If there is no

XtNmodifyVerification callback or if the *ok* field of the OlTextModifyCallData structure is TRUE, then the text will be updated. Then the activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection and cursor position before the last modification. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor and selection will be updated. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure. Finally, the XtNpostModifyNotification callback list will be called with the appropriate OlTextPostModifyCallData structure.

OL_WORDBAK

The OL_WORDBAK activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved one word before the current cursor position. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved. When the cursor is positioned before the first word on a line, the previous line will be brought into view and the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_WORDFWD

The OL_WORDFWD activation type first calls the XtNkeys callback list with the appropriate OlInputCallData structure. If there is no XtNkeys callback or if the *consumed* field of the OlInputCallData structure is TRUE, then processing continues. The activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved one word after the current cursor position. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved. When the cursor is positioned after the last word on a line, the next line will be brought into view and the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

See Also

"StaticText Widget" on page 600,
"TextEdit Functions" on page 660,
"TextField Widget" on page 665,
"TextField Functions" on page 686,
"TextLine Widget" on page 688,
"TextLine Functions" on page 708,
"Text Selection Operations" on page 204.

10

TextEdit Functions

TextEdit Functions

The following functions assist in manipulating TextEdit widgets.

OlTextEditClearBuffer

```
#include <Xol/textbuff.h>

Boolean OlTextEditClearBuffer(
    TextEditWidget ctx);
```

`OlTextEditClearBuffer()` deletes all of the text associated with the TextEdit widget *ctx*. It returns FALSE if the widget supplied is not a TextEdit widget or if the clear operation fails; otherwise, it returns TRUE.

OlTextEditReadSubString

```
#include <Xol/textbuff.h>

Boolean OlTextEditReadSubString(
    TextEditWidget     ctx,
    char               **buffer,
    TextPosition       start,
    TextPosition       end);
```

`OlTextEditReadSubString()` retrieves a copy of a substring from the TextBuffer associated with the TextEdit widget. The storage required for the copy is allocated by this routine; it is the responsibility of the caller to free this storage when appropriate. It returns FALSE if the widget supplied is not a TextEdit widget or if the operation fails; otherwise, it returns TRUE.

OlTextEditCopyBuffer

```
#include <Xol/textbuff.h>

Boolean OlTextEditCopyBuffer(
    TextEditWidget     ctx,
    char               *buffer);
```

`OlTextEditCopyBuffer()` retrieves a copy of the contents of the TextBuffer associated with the TextEdit widget *ctx*. The storage required for the copy is allocated by this routine; it is the responsibility of the caller to free this storage when appropriate. The function returns FALSE if the widget supplied is not a TextEdit widget or if the buffer cannot be read; otherwise, it returns TRUE.

When `OlTextEditCopyBuffer()` returns, *buffer* contains the text in the TextBuffer of the widget in an (`OlStr *`); the `char **` shown in the synopsis is for binary compatibility with previous versions.

OlTextEditCopySelection

```
#include <Xol/textbuff.h>

Boolean OlTextEditCopySelection(
    TextEditWidget      ctx,
    int                 delete);
```

`OlTextEditCopySelection()` Copies or Cuts the current selection in the TextEdit widget *ctx*. If no selection exists, or if the TextEdit cannot acquire the CLIPBOARD, or if the widget supplied is not a TextEdit widget, FALSE is returned. Otherwise, the selection is copied to the CLIPBOARD then, if the *delete* flag is nonzero, the text is then deleted from the TextBuffer associated with the TextEdit widget (i.e., a Cut operation is performed). Finally, TRUE is returned.

OlTextEditRedraw

```
#include <Xol/textbuff.h>

Boolean OlTextEditRedraw(
    TextEditWidget ctx);
```

`OlTextEditRedraw()` forces a complete refresh of the TextEdit widget display. It returns FALSE if the widget supplied is not a TextEdit widget or if the widget is not realized or if the update state is FALSE; otherwise, it returns TRUE.

OlTextEditGetCursorPosition

```
#include <Xol/textbuff.h>

Boolean OlTextEditGetCursorPosition(
    TextEditWidget      ctx,
    TextPosition        *start,
    TextPosition        *end,
    TextPosition        *cursorPosition);
```

`OlTextEditGetCursorPosition()` retrieves the current selection *start*, *end*, and *cursorPosition*. The *start* value will represent the position of the first character in the selection; the *end* value will be the position of the character

after the last character in the selection. (For example, if the TextBuffer contains `abc` and the selection is `ab`, *start* will return as 0 and *end* as 2.) If there is no current selection, *start* and *end* will both be equal to *cursorPosition*. The function returns FALSE if the widget supplied is not a TextEdit widget; otherwise, it returns TRUE.

OlTextEditSetCursorPosition

```
#include <Xol/textbuff.h>

Boolean OlTextEditSetCursorPosition(
    TextEditWidget      ctx,
    TextPosition        start,
    TextPosition        end,
    TextPosition        cursorPosition);
```

`OlTextEditSetCursorPosition()` changes the current selection *start* and *end* and *cursorPosition*. For efficiency, the function does not check the validity of the positions. If invalid values are given, results are unpredictable. The function attempts to ensure that the *cursorPosition* is visible by scrolling the display. It returns FALSE if the widget supplied is not a TextEdit widget; otherwise, it returns TRUE.

OlTextEditGetLastPosition

```
#include <Xol/textbuff.h>

Boolean OlTextEditGetLastPosition(
    TextEditWidget      ctx,
    TextPosition        *position);
```

`OlTextEditGetLastPosition()` retrieves the TextEdit widget *ctx*. It returns FALSE if the widget supplied is not a TextEdit widget; otherwise, it returns TRUE.

OlTextEditMoveDisplayPosition

```
#include <Xol/textbuff.h>

void OlTextEditMoveDisplayPosition(
    TextEditWidget      ctx,
    OlInputEvent        move_type);
```

`OlTextEditMoveDisplayPosition()` moves the display position and performs scroll updates in the TextEdit widget. It is recommended that this function be used instead of `XtSetValues()` to move the display position. Using `XtSetValues()` may result in a screen flicker.
The *move_type* parameter can have one of the following values:

OL_SCROLLUP	Scroll up a line
OL_SCROLLDOWN	Scroll down a line
OL_PAGEUP	Scroll a page up
OL_PAGEDOWN	Scroll a page down
OL_HOME	Scroll to the beginning of the text(home)
OL_END	Scroll to the end

OlTextEditTextBuffer

```
#include <Xol/textbuff.h>

TextBuffer *OlTextEditTextBuffer(
    TextEditWidget ctx);
```

`OlTextEditTextBuffer()` retrieves the TextBuffer pointer associated with the TextEdit widget *ctx*. This pointer can be used to access the facilities provided by the functions in "Text Buffer Functions" on page 163 and "Text Buffer Functions for Internationalization" on page 176. The function returns NULL if the text format of the *ctx* widget is not single-byte.

OlTextEditOlTextBuffer

```
#include <Oltextbuff.h>

OlTextBufferPtr OlTextEditOlTextBuffer(
    TextEditWidget     ctx);
```

`OlTextEditOlTextBuffer()` retrieves the `OlTextBufferPtr` associated with the TextEdit widget *ctx*. `OlTextBufferPtr` is an opaque pointer that points to a TextBuffer that is capable of handling multibyte and wide character data. This type of text buffer is only associated with a multibyte (OL_MB_STR_REP) or wide character (OL_WC_STR_REP) TextEdit widget.

The function returns NULL if the text format of the *ctx* widget is single-byte. It returns an `OlTextBufferPtr` if the text format of *ctx* is multibyte or wide character. For a single-byte (OL_SB_STR_REP) TextEdit widget, use `OlTextEditTextBuffer()`.

OlTextEditInsert

```
#include <Xol/textbuff.h>

Boolean OlTextEditInsert(
    TextEditWidget      ctx,
    String              buffer,
    int                 length);
```

`OlTextEditInsert()` inserts a NULL-terminated *buffer* containing *length* bytes in the TextBuffer associated with the TextEdit widget *ctx*. The inserted text replaces the current (if any) selection. The value of *length* is not used internally, but is passed on as the length field in the `XtNmodifyVerification` callback. The function returns FALSE if the widget supplied is not a TextEdit widget or if the insert operation fails; otherwise, it returns TRUE.

OlTextEditUpdate

```
#include <Xol/textbuff.h>

Boolean OlTextEditUpdate(
    TextEditWidget      ctx,
    Boolean             state);
```

`OlTextEditUpdate()` sets the *updateState* of a TextEdit widget. Setting the state to FALSE turns screen update off; setting the state to TRUE turns screen updates on and refreshes the display. The function returns FALSE if the widget supplied is not a TextEdit widget; otherwise, it returns TRUE.

OlTextEditPaste

```
#include <Xol/textbuff.h>

Boolean OlTextEditPaste(
    TextEditWidget ctx);
```

`OlTextEditPaste()` pastes the contents of the CLIPBOARD into the TextEdit widget *ctx*. The current (if any) selection is replaced by the contents of the CLIPBOARD. The function returns FALSE if the widget supplied is not a TextEdit widget; otherwise, it returns TRUE.

See Also

Regular Expression Functions on page 161,
"Text Buffer Functions" on page 163,
"Text Buffer Functions for Internationalization" on page 176.

TextField Widget

Note – The TextField widget is obsolete but remains in the toolkit for backward compatibility. Its functionality has been superseded by the TextLine widget, which provides a more efficient implementation of a one-line text field. See "TextLine Widget" on page 688 for more information.

Class

Class Name: `TextField`
Class Pointer: `textFieldWidgetClass`

Ancestry

Core-Composite-Constraint-Manager-TextField

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/TextField.h>
```

Description

Components

A TextField widget is a one-line input field for text data, as shown in the following diagram.

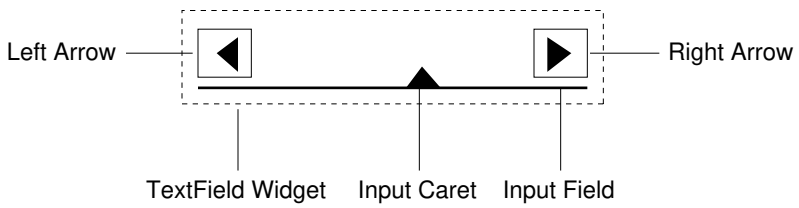

Figure 10-3 TextField Components

The input field and input caret are always present; however, the left and right arrows only appear when the value of the TextField is larger than the input field can display. These arrows provide the user the ability to scroll the TextField value.

Subwidget

The TextField widget creates a one-line TextEdit widget to handle the text editing. If the application needs to configure certain TextEdit resources (i.e., `XtNcursorPosition`), it can access the handle to the TextEdit subwidget through the `XtNtextEditWidget` resource. For more information on the TextEdit widget, see "TextEdit Widget" on page 623.

Keyboard Input

Once the input focus has been moved to the Input Field, keyboard entry is allowed. The TextField widget does not validate the input, leaving that up to the application.

Coloration

For 3D and 2D, `XtNfontColor` is used to draw the TextField's text and `XtNinputFocusColor` is used to draw the active caret.

For 3D, the TextField underline and scrollbutton coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1.

For 2D, `XtNbackground` and `XtNforeground` are used to render the TextField's underline and scrollbuttons as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

Keyboard Traversal

The default value of the `XtNtraversalOn` resource is TRUE.

The TextField widget responds to the following Activation Types for keyboard navigation. For more information on these Activation Types and their key bindings, see Table 10-9 on page 675.

OL_CHARBAK	OL_MENU	OL_PREVFIELD
OL_CHARFWD	OL_MENUKEY	OL_WORDBAK
OL_LINEEND	OL_NEXTFIELD	OL_WORDFWD
OL_LINESTART		

Keyboard Mnemonic Display

The TextField does not display the mnemonic. If the TextField is the child of a Caption widget, the Caption widget can be used to display the mnemonic.

Keyboard Accelerator Display

The TextField does not respond to a keyboard accelerator because clicking the SELECT button on a TextField activates depending on the pointer position. So, the TextField does not display a keyboard accelerator.

Text Selection

The TextField widget responds to the following Activation Types for selection. For more information on these Activation Types and their key bindings, see Table 10-9 on page 675.

OL_ADJUST	OL_SELCHARBAK	OL_SELLINE
OL_ADJUSTKEY	OL_SELCHARFWD	OL_SELLINEBAK
OL_COPY	OL_SELECT	OL_SELLINEFWD
OL_CUT	OL_SELECTKEY	OL_SELWORDBAK
OL_PASTE	OL_SELFLIPENDS	OL_SELWORDFWD

Scrolling

The TextField widget responds to the following Activation Types for scrolling. For more information on these Activation Types and their key bindings, see Table 10-9 on page 675.

OL_SCROLLRIGHT	OL_SCROLLLEFT	OL_SCROLLLEFTEDGE
OL_SCROLLRIGHTEDGE		

Editing

The TextField widget responds to the following Activation Types for editing. For more information on these Activation Types and their key bindings, see Table 10-9 on page 675.

OL_DELCHARBAK	OL_DELLINEBAK	OL_DELWORDFWD
OL_DELCHARFWD	OL_DELLINEFWD	OL_RETURN
OL_DELLINE	OL_DELWORDBAK	OL_UNDO

Scrolling Long Text Input

If an input value exceeds the length of the Input Field, the Left Arrow and/or Right Arrow appear and the input value is visually truncated on the left and/or the right to show only as many characters as can fit in the Input Field. The truncation is at a character boundary. Since the Arrows take up space that would otherwise be used for the input, the truncation is more severe than would be necessary if they were not visible. An Arrow is present only if characters are hidden in the direction expressed by the arrow.

The user can scroll to show the hidden parts of the input by:

- Clicking or pressing SELECT on the Left or Right Arrow.
- Clicking SELECT on the Left Arrow scrolls the input one character to the right to show the next character that was hidden to the left.
- Clicking SELECT on the Right Arrow scrolls the input one character to the left to show the next character that was hidden to the right.
- Pressing SELECT scrolls continuously, with a user-adjustable wait between changes.

The text does not scroll beyond its limits, so that the left-most character never moves beyond the right edge of the TextField widget and the rightmost character never moves beyond the left edge.

- If the user attempts to scroll beyond the limits by clicking SELECT, the system beeps.
- If the user is pressing SELECT when the limit is reached, the text stops scrolling but the system does not beep.
- If the user releases SELECT and then presses it again to exceed the scrolling limit, the system beeps once regardless of how long SELECT is pressed.

Input Validation

A validation callback list can be used to perform limited per-field validation, such as when the user presses the RETURN, PREVFIELD, or NEXTFIELD keys. It is not called if the user moves the focus to another input area using the mouse.

Caret Position

As characters are entered from the keyboard, the input caret moves to the right until it reaches the right end of the input field. As additional characters are typed the text scrolls to the left (the left arrow appears as discussed above) and the input caret moves relative to the text but remains stationary on the screen.

Selecting and Operating on the Input Field

The TextField widget allows text to be copied or moved to and from the input field. See "Text Selection Operations" on page 204 for the description of these operations.

Resources

Table 10-5 TextField Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 10-6 TextField Composite Resources

Name	Type	Default	Access
XtNchildren	WidgetList	NULL	G
XtNinsertPosition	XtOrderProc	NULL	SGI
XtNnumChildren	Cardinal	0	G

Table 10-7 TextField Manager Resources

Name	Type	Default	Access
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNinputFocusColor	Pixel	Red	SGID
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNunrealizeCallback	XtCallbackList	NULL	SGIO
XtNuserData	XtPointer	NULL	SGI

Table 10-8 TextField Resources

Name	Type	Default	Access
XtNcharsVisible	int	(calculated)	GI
XtNeditType	OlEditType	OL_TEXT_EDIT	SGI
XtNfont	OlFont	XtDefaultFont	SGI
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNimPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI
XtNinitialDelay	int	500	SGI
XtNinsertTab	Boolean	FALSE	SGI
XtNmaximumSize	int	0	SGI
XtNrepeatRate	int	100	SGI
XtNscale	int	12	SGI
XtNstring	OlStr	NULL	SGI
XtNtextEditWidget	Widget	NULL	G
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNverification	XtCallbackList	NULL	SGIO

XtNcharsVisible

Class	Type	Default	Access
XtCCharsVisible	int	(calculated)	GI

Synopsis: The initial width of the list in terms of characters.

This resource overrides the `XtNwidth` resource setting. The `XtNwidth` is recalculated to be the value of the average font width plus the values for the internal left and right margins. The value of this resource changes to reflect the effects of geometry changes imposed by the widget tree and the user. Calls to `XtSetValues()` for this resource are ignored.

XtNeditType

Class	Type	Default	Access
XtCEditType	OlEditMode	OL_TEXT_EDIT	SGI

Synopsis: The edit state of the source.
Values: `OL_TEXT_READ/"textread"` – The source is read-only; the user cannot edit it.
`OL_TEXT_EDIT/"textedit"` – The source is fully editable.

XtNfont

Class	Type	Default	Access
XtCFont	OlFont	XtDefaultFont	SGI

The TextField widget supports this resource in the same manner as a widget that would inherit it from the Primitive class. See "XtNfont" on page 26.

XtNfontColor

Class	Type	Default	Access
XtCFontColor	Pixel	XtDefaultForeground	SGID

The TextField widget supports this resource in the same manner as a widget that would inherit it from the Primitive class. See "XtNfontColor" on page 27.

XtNimPreeditStyle

Class	Type	Default	Access
XtCImPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI

Synopsis: The pre-edit style (in conjunction with the shell's

`XtNimStatusStyle` resource). If the pre-edit style is not supported by the input method, the ability to pre-edit is lost.

Values: `OL_ON_THE_SPOT/"onTheSpot"` – The pre-edit data is displayed at the insertion point in the application window. The preexisting user data is shifted and the pre-edit data is inserted at the point of insertion.
`OL_OVER_THE_SPOT/"overTheSpot"` – The pre-edit data is displayed in the application window, starting at the insertion point. As the user types the pre-edit data, the preexisting user data is obscured by the pre-edit data.
`OL_ROOT_WINDOW/"rootWindow"` – The pre-edit data is displayed in a child of the root window, away from the point of insertion.
`OL_NO_PREEDIT/"none"` – No pre-edit data is displayed.

See "XtNimStatusStyle" on page 44 and "Setting the Input Method Pre-Edit and Status Styles (Asian Locales Only)" on page 82.

XtNinitialDelay

Class	Type	Default	Access
XtCInitialDelay	int	500	SGI

Synopsis: The number of milliseconds from the time the scrolling arrows are pressed until the repeated scrolling starts.

XtNinsertTab

Class	Type	Default	Access
XtCInsertTab	Boolean	FALSE	SGI

Synopsis: Determines whether a tab character is insertable into the text.
Values: `TRUE/"true"` – Tabs are insertable but not used for forward traversal. Forward traversal can still be accomplished with Control-Tab.
`FALSE/"false"` – A tab is not insertable.

Setting this resource to FALSE (the default) makes traversal of the controls easier if the tab key is bound as OL_NEXTFIELD.

XtNmaximumSize

Class	Type	Default	Access
XtCLength	int	0	SGI

Synopsis: The maximum number of characters that can be entered into the internal buffer.

Values: `0 ≤ XtNmaximumSize`

If this value is not set or is zero, the internal buffer will increase its size as needed, limited only by memory limitations of the process.

XtNrepeatRate

Class	Type	Default	Access
XtCRepeatRate	int	100	SGI

Synopsis: The time in milliseconds between repeats when the scrolling arrows are pressed for more than `XtNinitialDelay` milliseconds.

XtNscale

Class	Type	Default	Access
XtCScale	int	12	SGI

The TextField widget supports this resource in the same manner as a widget that would inherit it from the Primitive class. See “XtNscale” on page 29.

XtNstring

Class	Type	Default	Access
XtCString	OlStr	NULL	SGI

Synopsis: The content of the Input Field. On being set, a copy of the value is made in an internal buffer.

Values: Any `OlStr` value valid in the current locale.

Using `XtGetValues()` on this resource gets a new copy that the application is responsible for freeing when no longer needed.

The TextField function `OlTextFieldGetOlString()` can also be used to get a copy of the `XtNstring` value; see page 686.

XtNtextEditWidget

Class	Type	Default	Access
XtCTextEditWidget	Widget	NULL	G

Synopsis: The TextEdit widget managed by the TextField.

This value can be used to directly access the underlying TextEdit widget (and its TextBuffer) used to manage the textual display.

XtNtextFormat

Class	Type	Default	Access
XtCTextFormat	OlStrRep	OL_SB_STR_REP	GI

The TextField widget supports this resource in the same manner as a widget that would inherit it from the Primitive class. See "XtNtextFormat" on page 29.

XtNverification

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the user presses the RETURN, PREVFIELD, or NEXTFIELD keys out of the TextField widget.

The *call_data* parameter is a pointer to a `OlTextFieldVerify` structure:

```
typedef struct _OlTextFieldVerify {
    String                  string;
    Boolean                 ok;
    OlTextVerifyReason      reason;
} OlTextFieldVerify;
```

string A pointer to the content of the text field. It is not a copy but a pointer to an internal buffer. The application should copy the buffer if it needs to keep the data intact longer than the duration of the callback.

ok Currently unused.

reason One of the following constants:

`OlTextFieldReturn`	RETURN entered
`OlTextFieldPrevious`	PREVFIELD entered
`OlTextFieldNext`	NEXTFIELD entered

Activation Types

The following table lists the activation types used by the TextField.

Table 10-9 TextField Activation Types

Activation Type	Semantics	Resource Name
OL_ADJUST	ADJUST	XtNadjustBtn
OL_ADJUSTKEY	ADJUST	XtNadjustKey
OL_CANCEL	CANCEL	XtNcancelKey
OL_CHARBAK	LEFT	XtNleftKey
OL_CHARFWD	RIGHT	XtNrightKey
OL_COPY	COPY	XtNcopyBtn
OL_CUT	CUT	XtNcutBtn
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_DELCHARBAK	DELETE BACKWARD	XtNdelCharBakFwd
OL_DELCHARFWD	DELETE FORWARD	XtNdelCharFwdKey
OL_DELLINE	DELETE LINE	XtNdelLineKey
OL_DELLINEBAK	DELLINEBAK	XtNdelLineBakKey
OL_DELLINEFWD	DELLINEFWD	XtNdelLineFwdKey
OL_DELWORDBAK	DELWORDBAK	XtNdelWordBakKey
OL_DELWORDFWD	DELWORDFWD	XtNdelWordFwdKey
OL_HELP	HELP	XtNhelpKey
OL_LINEEND	ROW END	XtNlineEndKey
OL_LINESTART	ROW START	XtNlineStartKey
OL_MENU	MENU	XtNmenuBtn
OL_MENUKEY	MENU	XtNmenuKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PASTE	PASTE	XtNpasteBtn
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_RETURN	RETURN	XtNreturnKey
OL_SCROLLLEFT	SCROLLLEFT	XtNscrollLeftKey
OL_SCROLLLEFTEDGE	SCROLLLEFTEDGE	XtNscrollLeftEdgeKey

Table 10-9 TextField Activation Types (Continued)

Activation Type	Semantics	Resource Name
OL_SCROLLRIGHT	SCROLLRIGHT	XtNscrollRightKey
OL_SCROLLRIGHTEDGE	SCROLLRIGHTEDGE	XtNscrollRightEdgeKey
OL_SELCHARBAK	SELCHARBAK	XtNselCharBakKey
OL_SELCHARFWD	SELCHARFWD	XtNselCharFwdKey
OL_SELECT	SELECT	XtNselectBtn
OL_SELECTKEY	SELECT	XtNselectKey
OL_SELFLIPENDS	SELFLIPENDS	XtNselFlipEndsKey
OL_SELLINE	SELLINE	XtNselLineKey
OL_SELLINEBAK	SELLINEBAK	XtNselLineBakKey
OL_SELLINEFWD	SELLINEFWD	XtNselLineFwdKey
OL_SELWORDBAK	SELWORDBAK	XtNselWordBakKey
OL_SELWORDFWD	SELWORDFWD	XtNselWordFwdKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey
OL_UNDO	UNDO	XtNundoKey
OL_WORDBAK	JUMP LEFT	XtNwordBakKey
OL_WORDFWD	JUMP RIGHT	XtNwordFwdKey

Activation types not described in the following table are described in "Common Activation Types" on page 68.

OL_ADJUST

The OL_ADJUST activation type calls the `XtNmotionVerification` callback list with the appropriate `OlTextMotionCallData` structure containing the new cursor position, selection start, and selection end. If a callback is not registered or the `OlTextMotionCallData`'s *ok* field is TRUE, the OL_ADJUST activation type will modify the `XtNcursorPosition`, `XtNselectStart`, and `XtNselectEnd` resources to reflect the cursor position and the resulting selection.

OL_CHARBAK

The OL_CHARBAK activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved one character before the current cursor position. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved.

OL_CHARFWD

The OL_CHARFWD activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved one character after the current cursor position. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved.

OL_COPY

The OL_COPY activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure containing the currently selected items. If the `OlTextModifyCallData` *ok* field is TRUE upon return, then the selected text will be copied to the CLIPBOARD.

OL_CUT

The OL_CUT activation type applies only to TextField widgets that have `XtNeditType` of OL_TEXT_EDIT. The `XtNmodifyVerification` callback list will be called with an `OlTextModifyCallData` structure containing the currently selected items. If the `OlTextModifyCallData` *ok* field is TRUE upon return, then the selected text will be copied to the CLIPBOARD.

OL_DELCHARBAK

The OL_DELCHARBAK activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the character to the left of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the character has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELCHARFWD

The OL_DELCHARFWD activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the character to the right of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the character has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELLINE

The OL_DELLINE activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the current line has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the line has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELLINEBAK

The OL_DELLINEBAK activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the line to the left of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification`

callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the line has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELLINEFWD

The OL_DELLINEFWD activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the line to the right of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the line has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELWORDBAK

The OL_DELWORDBAK activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the word to the left of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the word has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_DELWORDFWD

The OL_DELWORDFWD activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text after the word to the right of the cursor has been deleted. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position after the word has been deleted. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_LINEEND

The OL_LINEEND activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved to after the last character on the current line. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved.

OL_LINESTART

The OL_LINESTART activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved to before the first character on the current line. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved.

OL_MENU/ OL_MENUKEY

The OL_MENU and OL_MENUKEY activation types will pop up the TextField menu.

OL_PASTE

The OL_PASTE activation type applies only to TextField widgets that have `XtNeditType` of OL_TEXT_EDIT. When the widget has focus, then the OL_PASTE activation type will insert the contents of the CLIPBOARD at the current insert position by calling the `XtNpostModifyVerification` callback list. The `XtNmotionVerification` callback list will be called with an `OlTextMotionCallData` structure that represents the new cursor and selection position.

OL_RETURN

This activation type calls the `XtNverification` callback list with the appropriate `OlTextFieldVerify` structure.

OL_SCROLLLEFT

The OL_SCROLLLEFT activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the character left of the current view into the view.

OL_SCROLLLEFTEDGE

The OL_SCROLLLEFTEDGE activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the left-most character of the document into the view.

OL_SCROLLRIGHT

The OL_SCROLLRIGHT activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the character right of the current view into the view.

OL_SCROLLRIGHTEDGE

The OL_SCROLLRIGHTEDGE activation type calls the `XtNmargin` callback list with the appropriate `OlTextMarginCallData` structure to bring the rightmost character of the document into the view.

OL_SELCHARBAK

The OL_SELCHARBAK activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended one character to the left of the cursor position and the cursor position moved to before the selection. If there is no

XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, cursor will be moved, and the text may be scrolled to bring the previous line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELCHARFWD

The OL_SELCHARFWD activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended one character to the right of the cursor position and the cursor position moved to after the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, cursor will be moved, and the text may be scrolled to bring the next line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELECT

The OL_SELECT activation type calls the XtNmotionVerification callback list with the appropriate OlTextMotionCallData structure containing the new cursor position, selection start, and selection end. If a callback is not registered or the OlTextMotionCallData's *ok* field is TRUE, the OL_SELECT activation type will modify the XtNcursorPosition, XtNselectStart, and XtNselectEnd resources to reflect the cursor position and the resulting selection.

OL_SELFLIPENDS

The OL_SELFLIPENDS activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the cursor position moved to the opposite end of the current selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the cursor will be moved. If the text is scrolled to bring the cursor into view, the XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELLINE

The OL_SELLINE activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended to the beginning of the current line and to the end of the current line.

and the cursor position moved to before the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELLINEBAK

The OL_SELLINEBAK activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended to the beginning of the line to the left of the current cursor position and the cursor position moved to before the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, cursor will be moved, and the text may be scrolled to bring the previous line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELLINEFWD

The OL_SELLINEFWD activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended to the end of the line to the right of the current cursor position and the cursor position moved to after the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, cursor will be moved, and the text may be scrolled to bring the next line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELWORDBAK

The OL_SELWORDBAK activation type calls the XtNmotionVerification callback list with an OlTextMotionCallData structure that represents the selection extended to the beginning of the word to the left of the current cursor position and the cursor position moved to before the selection. If there is no XtNmotionVerification callback or if the *ok* field of the OlTextMotionCallData structure is TRUE, then the selection will be extended, cursor will be moved, and the text may be scrolled to bring the previous line into view. The XtNmargin callback list will be called with the appropriate OlTextMarginCallData structure.

OL_SELWORDFWD

The OL_SELWORDFWD activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection extended to the end of the word to the right of the current cursor position and the cursor position moved to after the selection. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the selection will be extended, cursor will be moved, and the text may be scrolled to bring the next line into view. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure.

OL_UNDO

The OL_UNDO activation type calls the `XtNmodifyVerification` callback list with an `OlTextModifyCallData` structure that represents the selection, cursor position, and text before the last modification. If there is no `XtNmodifyVerification` callback or if the *ok* field of the `OlTextModifyCallData` structure is TRUE, then the text will be updated. Then the activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the selection and cursor position before the last modification. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor and selection will be updated. The `XtNmargin` callback list will be called with the appropriate `OlTextMarginCallData` structure. Finally, the `XtNpostModifyNotification` callback list will be called with the appropriate `OlTextPostModifyCallData` structure.

OL_WORDBAK

The OL_WORDBAK activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved one word before the current cursor position. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved.

OL_WORDFWD

The OL_WORDFWD activation type calls the `XtNmotionVerification` callback list with an `OlTextMotionCallData` structure that represents the cursor position moved one word after the current cursor position. If there is no `XtNmotionVerification` callback or if the *ok* field of the `OlTextMotionCallData` structure is TRUE, then the cursor will be moved.

See Also

"StaticText Widget" on page 600,
"TextEdit Widget" on page 623,
"TextEdit Functions" on page 660,
"TextField Functions" on page 686,
"Text Buffer Functions" on page 163,
"Text Selection Operations" on page 204.

10

TextField Functions

TextField Functions

The following functions assist in manipulating TextField widgets.

OlTextFieldCopyOlString

```
int OlTextFieldCopyOlString(
    TextFieldWidget    tfw,
    OlStr              string);
```

`OlTextFieldCopyOlString()` copies the `OlStr` string associated with the TextField widget *tfw* into the user-supplied area pointed to by *string*. The function returns the length of this string in number of bytes (if the text format is single-byte or multibyte) or in the number of wide characters (if the text format is wide character).

OlTextFieldCopyString

```
#include <textbuff.h>

int OlTextFieldCopyString(
    TextFieldWidget    tfw,
    char               *string);
```

`OlTextFieldCopyString()` copies the string associated with the `TextField` widget *tfw* into the user supplied area pointed to by *string* and returns the length of this string.

Note – This function is superseded by `OlTextFieldCopyOlString()`. However, it is safe to use `OlTextFieldCopyString()` if the text format of the widget is single-byte.

OlTextFieldGetOlString

```
OlStr OlTextFieldGetOlString(
    TextFieldWidget    tfw,
    int                *size);
```

`OlTextFieldGetOlString()` retrieves a new copy of the `OlStr` associated with the TextField widget *tfw*. The function returns a pointer to the newly allocated `OlStr` copy.

Optionally, if *size* is not NULL, `OlTextFieldGetOlString()` returns in *size* the length of the string in number of bytes (if the text format is single-byte or multibyte) or in number of wide characters (if the text format is wide character).

OlTextFieldGetString

```
#include <textbuff.h>

char *OlTextFieldGetString(
    TextFieldWidget    tfw,
    int                *size);
```

`OlTextFieldGetString()` retrieves a new copy of the string associated with the `TextField` widget *tfw* and returns a pointer to the newly allocated string copy. Optionally, if *size* is not NULL, the function returns in *size* the length of the string, including the null terminator.

Note – The storage for the copy is allocated by this routine. It is the responsibility of the caller to free this storage when it becomes dispensable.

Note – This function is superseded by `OlTextFieldGetString()`. However, it is safe to use `OlTextFieldGetString()` if the text format of the widget is single-byte.

See Also

Regular Expression Functions on page 161.

10

TextLine Widget

TextLine Widget

Class

Class Name: `TextLine`
Class Pointer: `textLineWidgetClass`

Ancestry

Core-Primitive-TextLine

Required Header Files

```
#include <Xol/OpenLook>
#include <Xol/TextLine.h>
```

Description

The TextLine widget is a one-line input field for text data. Once the input focus is moved into the widget, keyboard entry is allowed. If the input value exceeds the length of the input field, the scroll buttons appear. Hidden text can then be scrolled into view by pressing the scroll buttons. Pressing the buttons continuously scrolls the text repeatedly with a user-adjustable delay.

Components

The TextLine contains the following graphical elements:

- Right-justified bold label at the left of the TextLine
- Input field
- Input Caret (not present in ReadOnly mode)
- 1-point (for Mono) or chiseled underline (not present in ReadOnly mode)
- Optional scroll buttons

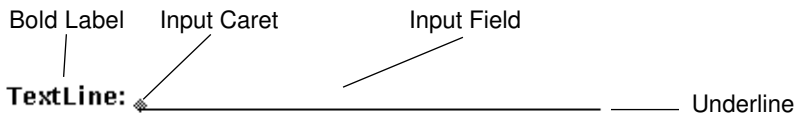

Figure 10-4 TextLine Components

Keyboard Traversal

The TextLine allows keyboard entry if it is sensitive and it has the keyboard focus. However, in ReadOnly mode, the widget is not traversable and it does not receive keyboard focus. Selection is also turned off during ReadOnly mode.

The TextLine responds to the following keyboard navigation keys:

- NEXTFIELD moves to the next traversable widget in the window
- PREVFIELD moves to the previous traversable widget in the window
- CHARFWD moves the caret forward one character
- CHARBAK moves the caret backward one character
- WORDFWD moves the caret forward one word
- WORDBAK moves the caret back one word
- LINESTART moves the caret to the beginning of the display
- LINEEND moves the caret to the end of the display
- MENUKEY posts the menu associated with the TextLine

The TextLine responds to the following edit keys:

- DELCHARFWD deletes the character to the right of the caret
- DELCHARBAK deletes the character to the left of the caret
- DELWORDFWD deletes the word to the right of the caret
- DELWORDBAK deletes the word to the left of the caret
- DELLINEFWD deletes to the end of the line from the caret
- DELLINEBAK deletes from the beginning of the line to the caret
- DELLINE deletes the line containing the caret
- UNDO undoes the last edit

Keyboard Mnemonic Display

The TextLine does not display any mnemonic. If the TextLine is the child of a Caption widget, the Caption can be used to display any mnemonic.

Keyboard Accelerator Display

The TextLine does not respond to any keyboard accelerators.

Display of Text

The TextLine displays its contents in the font specified by the `XtNfont` resource. If the length of the text exceeds the length of the input field, the widget sets up the Left or Right or both scroll buttons to indicate this. The text is then visually truncated at the boundaries to show only as many characters as can fit in the input field. The truncation is always at a character boundary. A scroll button is present only if characters are hidden in the direction indicated by that button. The user can scroll to show the hidden parts of the text by clicking or pressing the scroll buttons. Clicking SELECT on any scroll button will scroll the text one character in the direction indicated by that button. Pressing SELECT on any scroll button will repeat the scrolling with a user-adjustable delay between each scroll.

Caret Position

As characters are entered from the keyboard, the caret moves to the right until it reaches the right end of the input field. As additional characters are typed, the text jump-scrolls to the left by a specific amount. Note that the TextLine always keeps the cursor visible. Thus, the presence or absence of either of the scroll buttons is controlled by the current cursor position.

Selection of Text

Text selection can be done by the user by using the mouse or keyboard. The widget also provides a set of convenience functions to manipulate the selection programmatically; see “TextLine Functions” on page 708.

Coloration

For 3D and 2D, `XtNfontColor` is used to draw the TextLine’s text and `XtNinputFocusColor` is used to draw the active caret.

For 3D, the TextLine underline and scrollbutton coloration is defined by the *OPEN LOOK GUI Functional Specification*, Chapter 9, "Color and Three-Dimensional Design." `XtNbackground` is used for BG1, and the BG2 (pressed-in), BG3 (shadow), and Highlight colors are derived by the toolkit from BG1.

For 2D, `XtNbackground` and `XtNforeground` are used to render the TextLine's underline and scrollbuttons as described by the *OPEN LOOK GUI Functional Specification*, Chapter 4, "Controls."

Known Deficiencies

The TextLine widget currently does not support the *implicit commit* feature; see "Input Method" on page 80. This could be a deficiency in Asian locales. The workaround is to use the TextField widget (page 665), which does support it.

Resources

Table 10-10 TextLine Core Resources

Name	Type	Default	Access
XtNaccelerators	AcceleratorTable	NULL	SGI
XtNancestorSensitive	Boolean	TRUE	G
XtNbackground	Pixel	XtDefaultBackground	SGID
XtNbackgroundPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderColor	Pixel	XtDefaultForeground	SGID
XtNborderPixmap	Pixmap	XtUnspecifiedPixmap	SGI
XtNborderWidth	Dimension	1	SGI
XtNcolormap	Colormap	(parent's)	SGI
XtNdepth	int	(parent's)	GI
XtNdestroyCallback	XtCallbackList	NULL	SGIO
XtNheight	Dimension	(calculated)	SGI
XtNmappedWhenManaged	Boolean	TRUE	SGI
XtNscreen	Screen *	(parent's)	G
XtNsensitive	Boolean	TRUE	GIO
XtNtranslations	XtTranslations	NULL	SGI
XtNwidth	Dimension	(calculated)	SGI
XtNx	Position	0	SGI
XtNy	Position	0	SGI

Table 10-11 TextLine Primitive Resources

Name	Type	Default	Access
XtNaccelerator	String	NULL	n/a
XtNacceleratorText	String	NULL	n/a
XtNconsumeEvent	XtCallbackList	NULL	SGIO
XtNfont	OlFont	XtDefaultFont	SGI
XtNfontColor	Pixel	XtDefaultForeground	SGID
XtNforeground	Pixel	XtDefaultForeground	SGID
XtNinputFocusColor	Pixel	(calculated; see page 27)	SGID
XtNmnemonic	unsigned char	'\0'	n/a
XtNreferenceName	String	NULL	GI
XtNreferenceWidget	Widget	NULL	GI
XtNscale	int	12	SGI
XtNtextFormat	OlStrRep	OL_SB_STR_REP	GI
XtNtraversalOn	Boolean	TRUE	SGI
XtNuserData	XtPointer	NULL	SGI

Table 10-12 TextLine Resources

Name	Type	Default	Access
XtNblinkRate	int	1000	SGI
XtNcaptionAlignment[1]	OlDefine	OL_CENTER	G
XtNcaptionFont[1]	OlFont	OlDefaultBoldFont	SI
XtNcaptionLabel[1]	OlStr	NULL	SGI
XtNcaptionPosition[1]	OlDefine	OL_LEFT	G
XtNcaptionSpace[1]	Dimension	4	G
XtNcaptionWidth[1]	Dimension	0	G
XtNcharsVisible	int	0	GI
XtNcommitCallback	XtCallbackList	NULL	SGIO
XtNcursorPosition	int	0	SGI
XtNeditType	OlDefine	OL_TEXT_EDIT	SGI
XtNimPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI
XtNinitialDelay	int	500	SGI
XtNinsertTab	Boolean	FALSE	SGI

Table 10-12 TextLine Resources (Continued)

Name	Type	Default	Access
XtNmaximumChars	int	0	GI
XtNmenu	Widget	(special)	GI
XtNmotionCallback	XtCallbackList	NULL	SGIO
XtNpostModifyCallback	XtCallbackList	NULL	SGIO
XtNpreModifyCallback	XtCallbackList	NULL	SGIO
XtNrepeatRate	int	100	SGI
XtNstring	OlStr	NULL	SGI
XtNunderline	Boolean	TRUE	SGI
XtNupdateDisplay	Boolean	TRUE	SGI

1. These resources are provided to support captions. They are subject to change in a future OLIT release. Note that they cannot be set.

XtNblinkRate

Class	Type	Default	Access
XtCBlinkRate	int	1000	SGI

Synopsis: The blink rate of the active cursor in terms of milliseconds. A value of zero turns blinking off.
Values: Any integer

XtNcaptionAlignment

Class	Type	Default	Access
XtCCaptionAlignment	OlDefine	OL_CENTER	G

Synopsis: The alignment of the caption with respect to the text area.

Note – This resource cannot be set and is subject to change in future revisions.

XtNcaptionFont

Class	Type	Default	Access
XtCCaptionFont	OlFont	OlDefaultBoldFont	SI

Synopsis: The font for the caption label.
Values: Any font valid in the current locale

Note – This resource cannot be set and is subject to change in future revisions.

XtNcaptionLabel

Class	Type	Default	Access
XtCCaptionLabel	OlStr	NULL	SGI

Synopsis: The Label for the TextLine.
Values: Any `OlStr` valid in the current locale.

Note – This resource cannot be set and is subject to change in future revisions.

XtNcaptionPosition

Class	Type	Default	Access
XtCCaptionPosition	OlDefine	OL_LEFT	G

Synopsis: The position of the caption with respect to the text area.

Note – This resource cannot be set and is subject to change in future revisions.

XtNcaptionSpace

Class	Type	Default	Access
XtCCaptionSpace	Dimension	4	G

Synopsis: The separation between the caption and the text area in pixels.

Note – This resource cannot be set and is subject to change in future revisions.

XtNcaptionWidth

Class	Type	Default	Access
XtCCaptionWidth	Dimension	0	G

Synopsis: The width of caption text in pixels

Note – This resource cannot be set and is subject to change in future revisions.

XtNcharsVisible

Class	Type	Default	Access
XtCCharsVisible	int	0	GI

Synopsis: If nonzero, the initial width of text in terms of characters.
Values: Any integer

This resource overrides the `XtNwidth` setting. `XtNwidth` is then calculated as:

`XtNwidth` = `XtNcharsVisible` × *max_char_width over the given FontSet.*

The actual number of characters visible could be more than the value of this resource since `XtNwidth` is computed based on the *max_char_width* of the given `FontSet`. The resource value changes with Geometry changes.
If `XtNcharsVisible` is zero, the width of the text is determined by `XtNmaximumChars`. See page 697.

XtNcommitCallback

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when a <Tab> or <Return> is inserted into the TextLine.

The *call_data* structure is:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Boolean     valid;
    OlStr       buffer;
    int         length;
} OlTLCommitCallbackStruct;
```

Fields in the *call_data* structure are:

reason OL_REASON_COMMIT

event A pointer to the corresponding `XEvent` structure

valid A field to be set by the callback to indicate the validity of the contents

buffer A pointer to the `XtNstring` resource. This pointer is ReadOnly and valid only within the callback.

length Length of the text in characters, not including any terminating NULL character

This callback is invoked when the user hits the RETURN, NEXTFIELD, or PREVFIELD keys. If the callback sets *valid* to TRUE, the widget:

- Transfers focus to the next traversable widget if Mouseless mode is enabled.
- Transfers focus and insert point to the next TextLine or TextEdit if Mouseless is disabled.

If the callback sets *valid* to FALSE, the widget maintains focus and insert point in the current TextLine. The application also can provide additional feedback within this callback, such as popping up a Notice, clearing the field, or resetting the cursor position.

XtNcursorPosition

Class	Type	Default	Access
XtCTextPosition	int	0	SGI

Synopsis: The position of the cursor.
Values: 0 ≤ `XtNcursorPosition` ≤ *total number-of-characters*

Setting this resource will cause scrolling if the new position is beyond the visual area.

XtNeditType

Class	Type	Default	Access
XtCEditType	OlDefine	OL_TEXT_EDIT	SGI

Synopsis: The edit mode.
Values: `OL_TEXT_EDIT/"text_edit"` – The text is editable.
`OL_TEXT_READ/"text_read"` – The text is read-only. In read-only mode, the input-caret is disabled and the widget is not traversable or selectable.

XtNimPreeditStyle

Class	Type	Default	Access
XtCImPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI

Synopsis: The pre-edit style (in conjunction with the shell's `XtNimStatusStyle` resource). If the pre-edit style is not supported by the input method, the ability to pre-edit is lost.
Values: `OL_ON_THE_SPOT/"onTheSpot"` – The pre-edit data is displayed at the insertion point in the application window. The preexisting user data is shifted and the pre-edit data is inserted at the point of insertion.

`OL_OVER_THE_SPOT/"overTheSpot"` – The pre-edit data is displayed in the application window, starting at the insertion point. As the user types the pre-edit data, the preexisting user data is obscured by the pre-edit data.
`OL_ROOT_WINDOW/"rootWindow"` – The pre-edit data is displayed in a child of the root window, away from the point of insertion.
`OL_NO_PREEDIT/"none"` – No pre-edit data is displayed.

See "XtNimStatusStyle" on page 44 and "Setting the Input Method Pre-Edit and Status Styles (Asian Locales Only)" on page 82.

XtNinitialDelay

Class	Type	Default	Access
XtCInitialDelay	int	500	SGI

Synopsis: The time in milliseconds of the initial repeat delay to be used when the scrolling arrows are pressed.
Values: Any integer

XtNinsertTab

Class	Type	Default	Access
XtCInsertTab	Boolean	FALSE	SGI

Synopsis: Determines whether tabs are insertable.
Values: `TRUE/"true"` – Tabs are insertable but are not used for forward traversal. Forward traversal can still be accomplished with Control-Tab.
`FALSE/"false"` – Tabs are not insertable and act as NEXTFIELD.

XtNmaximumChars

Class	Type	Default	Access
XtCMaximumChars	int	0	GI

Synopsis: The maximum internal buffer size in terms of characters.
Values: Any integer

If this resource is zero, then the internal buffer will increase in size dynamically.

If `XtNcharsVisible` is zero (its default value), it is set up as follows:

```
if (charsVisible == 0)
    charsVisible = (maximumChars ? maximumChars : 20)
```

XtNmenu

Class	Type	Default	Access
XtCReadOnly	Widget	(special)	GI

Synopsis: The handle to the MenuShell widget that is popped up when the user presses the MENU key over the TextLine. By default, this menu contains the OPEN LOOK specified default elements UNDO, CUT, COPY, PASTE, and DELETE.

The application can augment this menu by creating more items as children of the MenuShell's `XtNmenuPane` widget. It can also change attributes of the MenuShell or the default items within. However, none of the default items should be removed from the MenuShell.

The application can also install its own Menu by setting this resource while creating the TextLine widget. In this case, the application's menu will be popped up instead of the widget's built-in menu. In fact, the widget does not even create its own menu. Therefore, an application can share menus among multiple TextLine widgets and avoid the overhead of having multiple MenuShells.

XtNmotionCallback

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked when the cursor position changes.

The *call_data* structure is:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Boolean     valid;
    int         current_cursor;
    int         new_cursor;
} OlTLMotionCallbackStruct;
```

The fields in the *call_data* structure are:

reason	OL_REASON_MOTION – Indicates that the callback was invoked due to non-programmatic cursor-movement. OL_REASON_PROG_MOTION – Indicates that this callback was invoked due to programmatic cursor-movement.
event	A pointer to the corresponding `XEvent` structure
valid	If TRUE, the widget performs the Cursor motion. Otherwise, it does not.
current_cursor	Current cursor position
new_cursor	New cursor position

XtNpreModifyCallback

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked before a modification of the buffer is attempted.

The *call_data* structure is:

```
typedef struct {
    int         reason;
    XEvent      *event;
    Boolean     valid;
    int         current_cursor;
    int         new_cursor;
    OlStr       buffer;
    int         start;
    int         replace_length;
    OlStr       insert_buffer;
    int         insert_length;
} OlTLPreModifyCallbackStruct;
```

The fields in the *call_data* structure are:

reason	OL_REASON_PRE_MODIFICATION – Indicates that the callback was invoked due to non-programmatic edits. OL_REASON_PROG_PRE_MODIFICATION – Indicates that this callback was invoked due to programmatic edits.
event	A pointer to the corresponding `XEvent` structure
valid	If TRUE, the widget performs the modification; otherwise, it does not

current_cursor	Current position of the cursor
new_cursor	Cursor position after modification
buffer	A pointer to the `XtNstring` resource. Note that this pointer is ReadOnly and will be valid only within the callback.
start	Start of text to be deleted or replaced
replace_length	Length in characters of the text to be deleted or replaced, not including any terminating NULL character.
insert_buffer	The new text to be inserted
insert_length	Length in characters of the text to be inserted, not including any terminating NULL character.

All fields except *valid* are ReadOnly.

XtNpostModifyCallback

Class	Type	Default	Access
XtCCallback	XtCallbackList	NULL	SGIO

Synopsis: The callback list invoked after modification of the buffer is done.

The *call_data* structure is:

```
typedef struct {
    int         reason;
    XEvent      *event;
    int         cursor;
    OlStr       buffer;
} OlTLPostModifyCallbackStruct;
```

Fields in the *call_data* structure are:

reason	OL_REASON_POST_MODIFICATION – Indicates that the callback was invoked due to non-programmatic edits. OL_REASON_PROG_POST_MODIFICATION – Indicates that this callback was invoked due to programmatic edits.
event	A pointer to the corresponding `XEvent` structure
cursor	Current position of cursor
buffer	A pointer to the `XtNstring` resource. This pointer is ReadOnly and will be valid only within the callback.

All fields are ReadOnly.

XtNrepeatRate

Class	Type	Default	Access
XtCRepeatRate	int	100	SGI

Synopsis: The time in milliseconds of the repeat delay to be used when the scrolling arrows are pressed.
Values: Any integer

XtNstring

Class	Type	Default	Access
XtCString	OlStr	NULL	SGI

Synopsis: The contents of the TextLine buffer.
Values: Any `OlStr` valid in the current locale.

On being set, the string is inserted into the widget's internal buffer. `XtGetValues()` on this resource returns a pointer to the current data. The widget should treat the pointed-to data as ReadOnly and the pointed-to data is guaranteed to be valid only until the next Intrinsics call. Thus, if the application needs the data longer, it should make a copy of it.

XtNunderline

Class	Type	Default	Access
XtCUnderline	Boolean	TRUE	SGI

Synopsis: The presence of the underline.
Values: `TRUE/"true"` – The underline is present.
`FALSE/"false"` – The underline is absent. Note that the *OPEN LOOK GUI Functional Specification* states that the underline should be removed if the widget is in read-only mode.

XtNupdateDisplay

Class	Type	Default	Access
XtCUpdateDisplay	Boolean	TRUE	SGI

Synopsis: The redisplay of the screen.
Values: `TRUE/"true"` – Redisplay the widget in the current state.
`FALSE/"false"` – Screen redisplay is stopped until it is set back to TRUE.

This resource is useful during incremental programmatic edits.

Activation Types

The following table lists the activation types used by the TextLine.

Table 10-13 TextLine Activation Types

Activation Type	Semantics	Resource Name
OL_CANCEL	CANCEL	XtNcancelKey
OL_CHARBAK	LEFT	XtNleftKey
OL_CHARFWD	RIGHT	XtNrightKey
OL_COPY	COPY	XtNcopyBtn
OL_CUT	CUT	XtNcutBtn
OL_DEFAULTACTION	DEFAULTACTION	XtNdefaultActionKey
OL_DELCHARBAK	DELETE BACKWARD	XtNdelCharBakFwd
OL_DELCHARFWD	DELETE FORWARD	XtNdelCharFwdKey
OL_DELLINE	DELETE LINE	XtNdelLineKey
OL_DELLINEBAK	DELLINEBAK	XtNdelLineBakKey
OL_DELLINEFWD	DELLINEFWD	XtNdelLineFwdKey
OL_DELWORDBAK	DELWORDBAK	XtNdelWordBakKey
OL_DELWORDFWD	DELWORDFWD	XtNdelWordFwdKey
OL_HELP	HELP	XtNhelpKey
OL_LINEEND	ROW END	XtNlineEndKey
OL_LINESTART	ROW START	XtNlineStartKey
OL_MOVEDOWN	MOVEDOWN	XtNdownKey
OL_MOVELEFT	MOVELEFT	XtNleftKey
OL_MOVERIGHT	MOVERIGHT	XtNrightKey
OL_MOVEUP	MOVEUP	XtNupKey
OL_NEXTFIELD	NEXTFIELD	XtNnextFieldKey
OL_PASTE	PASTE	XtNpasteBtn
OL_PREVFIELD	PREVFIELD	XtNprevFieldKey
OL_TOGGLEPUSHPIN	TOGGLEPUSHPIN	XtNtogglePushpinKey
OL_UNDO	UNDO	XtNundoKey
OL_WORDBAK	JUMP LEFT	XtNwordBakKey
OL_WORDFWD	JUMP RIGHT	XtNwordFwdKey

Activation types not described in the following table are described in "Common Activation Types" on page 68.

OL_CHARBAK

The cursor is moved backward by one character. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_CHARFWD

The cursor is moved forward by one character. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_COPY

This activation type copies the current selection from the widget to the CLIPBOARD.

OL_CUT

This activation type copies the current selection from the widget to the CLIPBOARD and also deletes the selected text from the widget. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELCHARBAK

If there exists a selection in the widget, it is deleted. If there is no selection, the character before the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.

`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELCHARFWD

If there exists a selection in the widget, it is deleted. If there is no selection, the character after the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELLINE

The whole line is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELLINEBAK

If there exists a selection in the widget, it is deleted. If there is no selection, the segment of the line before the insert point The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.

`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELLINEFWD

If there exists a selection in the widget, it is deleted. If there is no selection, the segment of the line after the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELWORDBAK

If there exists a selection in the widget, it is deleted. If there is no selection, the word before the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_DELWORDFWD

If there exists a selection in the widget, it is deleted. If there is no selection, the word after the insert point is deleted. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the deletion occurs. The callback can prevent the deletion by setting the *valid* field in the *call_data* to FALSE.

`XtNpostModifyCallback`	Invoked after the deletion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the deletion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_LINESTART

The cursor is moved to the start of the line. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_LINEEND

The cursor is moved to the end of the line. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_NEXTFIELD

This activation type invokes the `XtNcommitCallback`. If the callback sets *valid* to TRUE and if `XtNmouseless` is TRUE, the widget transfers focus to the next traversable widget. If the callback sets *valid* to TRUE and if `XtNmouseless` is FALSE, the widget transfers focus and sets the insert point to the next TextLine or TextEdit widget. If the callback sets *valid* to FALSE, the widget maintains focus and insert point within itself.

OL_PASTE

This activation type inserts the contents of the CLIPBOARD into the widget at the current insert point. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before the insertion occurs. The callback can prevent the insertion by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after the insertion occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the insertion. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_PREVFIELD

This activation type invokes the `XtNcommitCallback`. If the callback sets *valid* to TRUE and if `XtNmouseless` is TRUE, the widget transfers focus to the previous traversable widget. If the callback sets *valid* to TRUE and if `XtNmouseless` is FALSE, the widget transfers focus and sets the insert point to the previous TextLine or TextEdit widget. If the callback sets *valid* to FALSE, the widget maintains focus and insert point within itself.

OL_UNDO

This activation type undoes the last modification to the widget's text buffer. The following callbacks are invoked:

`XtNpreModifyCallback`	Invoked before any modification occurs. The callback can prevent the modification by setting the *valid* field in the *call_data* to FALSE.
`XtNpostModifyCallback`	Invoked after any modification occurs.
`XtNmotionCallback`	Invoked before the cursor position is changed due to the modification. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_WORDBAK

The cursor is moved backward by one word. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

OL_WORDFWD

The cursor is moved forward by one word. The `XtNmotionCallback` callback is invoked before the cursor position is changed. The callback can prevent the cursor movement by setting the *valid* field in the *call_data* to FALSE.

See Also

"NumericField Widget" on page 443,
"TextField Widget" on page 665,
"TextLine Functions" on page 708.

TextLine Functions

These functions manipulate the contents of the TextLine widget.

OlTLGetPosition

```
#include <TextLine.h>

int OlTLGetPosition(
    Widget  w,
    int     pos);
```

`OlTLGetPosition()` returns some key positions in the text line. Valid values for *pos* are:

OL_CURSORPOS	Return the cursor position.
OL_BEGIN_CURRENT_WORD	Return the beginning of the current word.
OL_END_CURRENT_WORD	Return the end of the current word.
OL_END_LINE	Return the end of the line.

`OlTLGetPosition()` returns the position corresponding to the specified value of *pos*. (No error information is available.)

OlTLGetSelection

```
#include <Xol/TextLine.h>

OlStr OlTLGetSelection(
    Widget  w,
    int     *start,
    int     *length);
```

`OlTLGetSelection()` returns the current Selection as well as the Selection *start* and *length*. The returned string should be freed by the application, when no longer required, using `XtFree()`. `OlTLGetSelection()` returns NULL if no selection is active.

OlTLGetSubString

```
#include <Xol/TextLine.h>

OlStr OlTLGetSubString(
    Widget  w,
    int     start,
    int     length);
```

`OlTLGetSubString()` returns *length* characters beginning at *start*. The returned string should be freed by the application using `XtFree()`. `OlTLGetSubString()` returns NULL on failure. Failures include invalid *start* and *length* values.

OlTLOperateOnSelection

```
#include <Xol/TextLine.h>

Boolean OlTLOperateOnSelection(
    Widget  w,
    int     mode);
```

`OlTLOperateOnSelection()` performs various operations on the primary or CLIPBOARD selections. Valid values for *mode* are:

OL_CUT	Copies the primary selection to the CLIPBOARD, then deletes the primary selection. This operation invokes the `XtNpreModifyCallback` and `XtNpostModifyCallback` callbacks.
OL_COPY	Copies the primary selection to the CLIPBOARD.
OL_PASTE	Inserts the CLIPBOARD selection at the destination cursor. If the destination cursor is inside the current selection, the CLIPBOARD selection replaces the selected text. This operation invokes the `XtNpreModifyCallback` and `XtNpostModifyCallback` callbacks.
OL_CANCEL	Clears the primary selection.

`OlTLOperateOnSelection()` returns FALSE if the primary selection is NULL, if the widget does not own the primary selection, or if the function is unable to gain ownership of the CLIPBOARD selection. Otherwise, it returns TRUE.

OlTLSetSelection

```
#include <Xol/TextLine.h>

Boolean OlTLSetSelection(
    Widget  w,
    int     start,
    int     length);
```

`OlTLSetSelection()` selects *length* characters beginning at *start*. It returns TRUE on success, FALSE on failure. Failures include invalid *start* and *length* values.

OlTLSetSubString

```
#include <Xol/TextLine.h>

Boolean OlTLSetSubString(
    Widget  w,
    int     start,
    int     length,
    OlStr   buffer);
```

`OlTLSetSubString()` can be used to do insertion, replacement, and deletion of substrings. It replaces *length* characters, beginning at *start*, with the contents of *buffer*. It returns TRUE on success, FALSE on failure. Failures include invalid *start* and *length* values and memory allocation failures.

See Also

"TextLine Widget" on page 688.