Complex Text Layout (CTL) Addendum

For Revision 2.1 of Motif *Programmer's Reference*, *User's Guide*, *Programmer's Guide*, and *Style Guide*

August 1997

Contents

Preface

Complex Text Layout (CTL) extensions enable Motif APIs to support languages whose writing systems require complex transformations between logical and physical representations of text, such as Arabic, Hebrew, and Thai. CTL Motif provides character shaping (such as ligatures, diacritics, and symmetrical swapping), and segment ordering. To support languages such as Arabic, a Motif API must include additional resources as described in Section 1.3.2.

The primary changes include:

- Widget behavior (see Section 1.4)
- New locales
- Runtime values of CTL-sensitive resources (see Section 1.4.2)

CTL Motif supports the complex text transformation of static and dynamic text widgets. It also supports right-to-left text, left-to-right text, and a combination of both (for dynamic and static text widgets), and tabbing for dynamic text widgets. Since text rendering is handled through the rendition layer, other widget libraries can be easily extended to support CTL.

To leverage the new CTL features, users must have the Portable Layout Services (PLS) library and the appropriate language engine. CTL uses PLS and the language engine to render text. Users must also specify the corresponding locale.

Bidirectional text causes the standard Motif single cursor to behave incorrectly in some situations. This sometimes leads to unintuitive behavior at boundaries (that is, where left-to-right and right-to-left text meets). A dual or split cursor seems to be a plausible solution, and will be addressed in a later CTL release.

1 Addendum to Motif *Programmer's Reference* (Revision 2.1)

1.1 XmDirection

1.1.1 Description

The XmNlayoutDirection resource[1] controls object layout. It interacts with the orientation value of the LayoutObject in the following manner.

First, when XmNlayoutDirection is specified as XmDEFAULT_DIRECTION, then the widget's layout direction is set at creation time from the governing pseudo-XOC. In the case of dynamic text (XmText and XmTextField), the governing pseudo-XOC is the one (if any) that is associated with the XmRendition used for the widget. In the case of static text (XmList, XmLabel, XmLabelG), the layout direction is set from the first compound string component that specifies a direction. This specification happens in one of two ways:

- Directly, if the component is of type XmSTRING_COMPONENT_LAYOUT_PUSH or XmSTRING_COMPONENT_DIRECTION, or
- Indirectly, if the component is of type XmSTRING_COMPONENT_LOCALE_TEXT, XmSTRING_COMPONENT_WIDECHAR_TEXT, or XmSTRING_COMPONENT_TEXT, from the component's associated XmRendition's associated LayoutObject.

Second, if XmNlayoutDirection is not specified as XmDEFAULT_DIRECTION, and the XmNlayoutModifier @ls orientation value is not specified explicitly in the layout modifier string, then the XmNlayoutDirection value is passed through to the XOC and its LayoutObject.

If both XmNlayoutDirection and the XmNlayoutModifier @ls orientation value are explicitly specified, then the behavior is mixed; the XmNlayoutDirection controls widget object layout, and the XmNlayoutModifier @ls orientation value controls layout transformations.

1.1.2 For More Information

For more information, see *CAE Specification: Portable Layout Services: Context-dependent and Directional Text,* The Open Group: Feb 1997; ISBN 1-85912-142-X; document number C616.

1.2 XmStringDirection

1.2.1 Description

XmStringDirection is the data type used to specify the direction in which the system displays characters of a string. The XmNlayoutDirection resource sets a default rendering direction for any compound string (XmString) that does not have a component specifying the direction of that string. Therefore, to set the layout direction, all that is required is to set the appropriate value for the XmNlayoutDirection resource. It is not required

[1] See section 11.3 of the Motif *Programmer's Guide* (Release 2.1) for an overview of XmNlayoutDirection, and especially for a description of the interaction between XmStringDirection and XmNlayoutDirection.

that you create compound strings with specific direction components. When the application renders an XmString, it should look to see if the string was created with an explicit direction (XmStringDirection). If there is no direction component, the application should check the value of the XmNlayoutDirection resource for the current widget and use that value as the default rendering direction for the XmString.

1.2.2 Related Information

See also XmRendition and XmDirection.

1.3 XmRendition

1.3.1 Description

1.3.2 New Resources

CTL adds the following new pseudo resources to XmRendition:

Name	Class/Type	Access	Default Value
XmNfontType	XmCFontType/XmFontType	CSG	XmAS_IS
XmNlayoutAttrObject	XmClayoutAttrObject/String	CG	NULL
XmNlayoutModifier	XmCLayoutModifier/String	CSG	NULL

XmNfontType

Specifies the type of the Rendition font object. For CTL, the value of this resource must be the XmFONT_IS_XOC value. If it is not, then the XmNlayoutAttrObject and XmNlayoutModifier resources are ignored.

When the value of this resource is XmFont_IS_XOC, and if the XmNfont resource is not specified, then at create time the value of the XmNfontName resource will be converted into an XOC object in either the locale specified by the XmNlayoutAttrObject resource or the current locale. Furthermore, the value of the XmNlayoutModifier resource will be passed through to any LayoutObject associated with the XOC.

XmNlayoutAttrObject

Specifies the layout AttrObject argument to be used to create the Layout Object associated with the XOC associated with this XmRendition. Refer to the Layout Services m_create_layout() specification for the syntax and semantics of this string. (See the description of XmNfontType above for an explanation of the interaction between the Layout Modifier Orientation output value and the XmNlayoutDirection widget resource).

XmNlayoutModifier

Specifies the layout values to be passed through to the Layout Object associated with the XOC associated with this XmRendition. For the syntax and semantics of this string, see *CAE Specification*.

Setting this resource via XmRendition{Retrieve,Update} causes the string to be passed through to the LayoutObject associated with the XOC associated with this Rendition. This is the mechanism for configuring layout services dynamically. Note that unpredictable behavior may result if the Orientation, Context, TypeOfText, TextShaping, or ShapeCharset are changed.

1.3.3 Additional Behavior

The XmNlayoutModifier affects the layout behavior of the text associated with the XmRendition. For example, if the layout default treatment of numerals is NUMERALS_NOMINAL, the user can change to NUMERALS_NATIONAL by setting XmNlayoutModifier to:

- @ls numerals=nominal:national, or
- @ls numerals=:national

The layout values can be classified into the following groups:

- Encoding description: TypeOfText, TextShaping, ShapeCharset (and locale codeset)

  TypeOfText is essentially segment ordering, and can be illustrated with opaque blocks. It is usually not meaningful to modify these values dynamically through the Rendition object, and will almost certainly result in unpredictable behavior.

- Layout behavior: Orientation, Context, ImplicitAlg, Swapping, Numerals

  Orientation and Context should not be modified dynamically; it is safe to modify ImplicitAlg, Swapping, and Numerals.

- Editing behavior: CheckMode

1.4 XmText, XmTextField

1.4.1 Description

Xm CTL extends XmText and XmTextField by adding a parallel set of movement and deletion actions that operate visually, patterned after the Motif 2.0 CSText widget. The standard Motif 2.1 Text and TextField do not distinguish between logical and physical order: “next“ and “forward“ mean “to the right,“ and “previous“ and “backward“ mean “to the left.“ CSText, however, makes the proper distinction and defines a new set of actions with strictly physical names (for example, left-character(), delete-right-word(), and so on). All of these action routines are defined to be sensitive to the XmNlayoutDirection of the widget and to call the appropriate “next-“ or “previous-“ action. The Xm CTL extensions are slightly more complex than CSText’s in that they are sensitive not to the global orientation of the widget, but to the specific directionality of the physical characters surrounding the cursor, as determined by the pseudo-XOC (including neutral stabilization).

There is also a new resource to control selection policy, to provide a rendition tag, and to control alignment.

The set of new Xm CTL actions is roughly the cross product of {Move,Delete,Kill} by {Left,Right} by {Character,Word}, and is listed below.

1.4.2 New Resources

The following new resources are added to XmText and XmTextField:

Name	Class/Type	Access	Default Value
XmNrenditionTag	XmCRenditionTag/XmRString	CSG	XmFONTLIST_DEFAULT_TAG
XmNalignment	XmCAlignment/XmRAlignment	CSG	XmALIGNMENT_BEGINNING
XmNeditPolicy	XmCEditPolicy/XmREditPolicy	CSG	XmEDIT_LOGICAL

XmNrenditionTag

Specifies the rendition tag of the XmRendition (in the XmNrenderTable resource) to be used for this widget.

XmNalignment

Specifies the text alignment to be used in the widget. Only XmALIGNMENT_END and XmALIGNMENT_CENTER are supported.

XmNeditPolicy

Specifies the editing policy to be used for the widget, either XmEDIT_LOGICAL or XmEDIT_VISUAL. In the case of XmEDIT_VISUAL, selection, cursor movement, and deletion will be in a visual style. Setting this resource also changes the translations for the standard keyboard movement and deletion events either to the new “visual“ actions list below or to the existing logical actions.

1.4.3 Action Routines

All of the actions in the following list query the orientation of the character(s) in the direction specified. If the direction is left-to-right, they call the corresponding next-/forward- or previous-/backward- variants.

- delete-left-character()
- delete-left-word()
- delete-right-character()
- delete-right-word()
- kill-left-character()
- kill-left-word()
- kill-right-character()
- kill-right-word()
- left-character()

- `left-word()`
- `right-character()`
- `right-word()`

1.4.4 Additional Behavior

The actions determine the orientation of characters by using the Layout Services transformation `OutToInp` and `Property` buffers (for the nesting level). The widget's behavior is therefore dependent on the locale-specific transformation. If the information in the `OutToInp` or, especially, `Property` buffers is inaccurate, the widget may behave unexpectedly. Moreover, as the locale-specific modules fall outside of the scope of this specification, bi-directional editing behavior may differ from platform to platform for the same text, application, resource values, and `LayoutObject` configuration.

The visual mode actions result in display cell-based behavior. The logical mode actions result in logical character-based behavior. For example, the `delete-right-character()` operation will delete the input buffer characters that correspond to the display cell (that is, one input buffer character whole `LayoutObject` transformation "property" byte "new cell indicator" is 1, and all of the succeeding characters whose "new cell indicator"[2] is 0).

Likewise, for `backward-character()`, the insertion point will be moved backward one character in the input buffer, and the cursor will be redrawn at the visual location corresponding to the associated output buffer character. This means that several keystrokes will be required to move across a composite display cell; the cursor will not actually change display location as the insertion point moves across input buffer characters whose "new cell indicator" is 0 (that is, diacritics or ligature fragments).

This means deletion operates either from the logical/input buffer side, or from the display cell level of the physical/output side. There is no mode for a strict, physical character-by-character deletion, since there is no one-to-one correspondence between the input and output buffers. A given physical character may represent only a fragment of a logical character, for example.

1.4.5 Action Routines

The `XmText` action routines are as follows:

`left-character(extend)`

If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and called without arguments, it moves the insertion cursor back logically by a character. If the insertion cursor is at the beginning of the line, it moves the insertion cursor to the logical last character of the previous line if exists, otherwise the insertion cursor position doesn't change.

If the `XmNeditPolicy` is `XmEDIT_VISUAL`, then the cursor moves to the left of cursor position. If the insertion cursor is at the beginning of the line, then it moves to the end character of the previous line if it exists.

If called with an argument of `extend`, it moves the insertion cursor as in the case of no argument and extends the current selection.

[2] For more information on the `Property` buffer, see the specification for `m_transform_layout()` in *CAE Specification*.

The left-character() action produces calls to the XmNmotionVerifyCallback procedures with the reason value XmCR_MOVING_INSERT_CURSOR. If called with an extend argument, this may produce calls to the XmNgainPrimaryCallback procedures. See the callback description in the Motif *Programmer's Reference* for more information.

left-word(extend)

If the XmNeditPolicy is XmEDIT_LOGICAL and is called without any arguments, and the insertion cursor is at the logical starting of the word, it moves the insertion cursor to the logical starting of the logical preceding word, if one exists, otherwise the insertion cursor position doesn't change. If the insertion cursor is in the word but not at the logical start of the word, it moves the insertion cursor to the logical start of the word. If the insertion cursor is at the logical start of the line, it moves the insertion cursor to the logical start of logical last word in the previous line if one exists, otherwise the insertion cursor position doesn't change.

If the XmNeditPolicy is XmEDIT_VISUAL and is called without arguments, it moves the insertion cursor to the first non-white space character after the first white space character to the left or after the beginning of the line. If the insertion cursor is already at the beginning of the word, it moves the insertion cursor to the beginning of the previous word. If the insertion cursor is already at the beginning of the line, it moves to the starting of the last word in the previous line.

If called with an argument of extend, it moves the insertion cursor as in the case of no argument and extends the current selection.

The left-word() action produces calls to the XmNmotionVerifyCallback procedures with the reason value XmCR_MOVING_INSERT_CURSOR. If it is called with an extend argument, this may produce calls to the XmNgainPrimaryCallback procedures. See the callback description in the Motif *Programmer's Reference* for more information.

right-character(extend)

If the XmNeditPolicy is XmEDIT_LOGICAL and is called without any arguments, it moves the insertion cursor logically forward by a character. If the insertion cursor is at the logical end of the line, it moves the insertion cursor to the logical starting of the next line, if one exists.

If the XmNeditPolicy is XmEDIT_VISUAL, then the cursor moves to the right of cursor position. If the insertion cursor is at the end of the line, it moves the insertion cursor to the starting of the next line, if one exists.

If called with an argument of extend, it moves the insertion cursor as in the case of no argument and extends the current selection.

The right-character() action produces calls to the XmNmotionVerifyCallback procedures with the reason value XmCR_MOVING_INSERT_CURSOR. If called with extend argument, this may produce calls to the XmNgainPrimaryCallback procedures. See the callback description in the Motif *Programmer's Reference* for more information.

right-word(extend)

If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and is called without any arguments, it moves the insertion cursor to the logical starting of the logical succeeding word if one exists, otherwise it moves to the logical end of the current word. If the insertion cursor is at the logical end of the line or in the logical last word of the line, it moves the cursor to the logical first word in the next line if one exists, otherwise it moves to the logical end of the current word.

If the `XmNeditPolicy` is `XmEDIT_VISUAL` and is called without arguments, it moves the insertion cursor to the first nonwhite space character after the first white space character to the right or after the end of the line.

If called with an argument of `extend`, it moves the insertion cursor as in the case of no argument and extends the current selection.

The `left-word()` action produces calls to the `XmNmotionVerifyCallback` procedures with the reason value `XmCR_MOVING_INSERT_CURSOR`. If called with `extend` argument, this may produce calls to the `XmNgainPrimaryCallback` procedures. See the callback description in the Motif *Programmer's Reference* for more information.

1.5 XmTextFieldGetLayoutModifier

1.5.1 Purpose

A `TextField` function that returns the layout modifier string which reflects the state of the layout object tied to its rendition.

1.5.2 Synopsis

`#include String XmTextFieldGetLayoutModifier(`*Widget widget*`)`

1.5.3 Description

`XmTextFieldGetLayoutModifier` accesses the value of the current layout object settings of the rendition associated with the widget. When the layout object modifier values are changed using a convenience function, the `XmTextFieldGetLayoutModifier` function will return the complete state of the layout object, not just the changed values.

1.5.4 Return Value

Returns the layout object modifier values in a form of a String value.

1.5.5 Related Information

`XmTextField`

1.6 XmTextGetLayoutModifier

1.6.1 Purpose

A `Text` function that returns the layout modifier string which reflects the state of the layout object tied to its rendition.

1.6.2 Synopsis

`#include String XmTextGetLayoutModifier(`*Widget widget*`)`

1.6.3 Description

`XmTextGetLayoutModifier` accesses the value of the current layout object settings of the rendition associated with the widget. When the layout object modifier values are changed using a convenience function, the `XmTextGetLayoutModifier` function will return the complete state of the layout object, not just the changed values.

1.6.4 Return Value

Returns the layout object modifier values in the form of a String value.

1.6.5 Related Information

`XmText`

1.7 XmTextFieldSetLayoutModifier

1.7.1 Purpose

A `TextField` function that sets the layout modifier values, which would change the behavior of the layout object tied to its rendition.

1.7.2 Synopsis

`#include String XmTextFieldSetLayoutModifier(`*Widget widget*`)`

1.7.3 Description

`XmTextFieldSetLayoutModifier` modifies the layout object settings of a rendition associated with the widget. When the layout object modifier values are set using this convenience function, only the attributes specified in the input parameter are changed; the rest of the attributes are left untouched.

1.7.4 Related Information

`XmTextField`

1.8 XmTextSetLayoutModifier

1.8.1 Purpose

A `Text` function that sets the layout modifier values, which would change the behavior of the layout object tied to its rendition.

1.8.2 Synopsis

`#include String XmTextSetLayoutModifier(`*Widget widget*`)`

1.8.3 Description

XmTextSetLayoutModifier modifies the layout object settings of a rendition associated with the widget. When the layout object modifier values are set using this convenience function, only the attributes specified in the input parameter are changed; the rest of the attributes are left untouched.

1.8.4 Related Information

XmText

1.9 XmStringDirectionCreate

1.9.1 Description

XmStringDirectionCreate creates a compound string with a single component, a direction with the given value. On the other hand, the XmNlayoutDirection resource sets a default rendering direction for any compound string (XmString) that does not have a component specifying the direction for that string. Therefore, to set the layout direction, all that is required is to set the appropriate value for the XmNlayoutDirection resource. It is not required to create compound strings with specific direction components. When the application renders an XmString, it should look to see if the string was created with an explicit direction (XmStringDirection). If there is no direction component, the application should check the value of the XmNlayoutDirection resource for the current widget and use that value as the default rendering direction for the XmString.

1.9.2 Related Information

See also XmRendition, XmDirection.

1.10 UIL

UIL Argument Name	Argument Type
XmNlayoutAttrObject	string
XmNlayoutModifier	string
XmNrenditionTag	string
XmNalignment	integer
XmNeditPolicy	integer

2 Addendum to Motif *User's Guide* (Revision 2.1)

2.1 Entering and Editing Text

Text components use an insertion cursor to indicate where the text you are typing will be inserted. When a Text component has the input focus, the insertion cursor is indicated by a blinking I-beam cursor.

To Move the Insertion Cursor

1. Move the pointer to the position where you want to begin typing.
2. Click SELECT (the left mouse button).

   The I beam insertion cursor will move to that location to let you know that you can begin typing. If a Text component has a input focus, you can navigate through the text using the keyboard. The left and right arrow keys move the insertion cursor by a character. The up and down arrow keys move the insertion cursor by a line. Pressing Control and using the left and right arrows navigates by words. Pressing Control and using the up and down arrow keys navigates by paragraphs. Page Up, Page Down, Page Left or Control-Page Up, and Page Right or Control-Page Down move the insertion cursor by pages.

To Select Text

Use any of the following methods to select text.

1. Press SELECT and drag over the region that you want to select.
2. Double-click SELECT in a word to select the word.
3. Click SELECT multiple times to select larger amounts of text as determined by the application.

The text selected will be dictated by the `XmNeditPolicy` resource which can be set to `XmEDIT_VISUAL` or `XmEDIT_LOGICAL`. If `XmNeditPolicy` is set to `XmEDIT_VISUAL`, the selection can be a contiguous region of text, even if the text contains bi-directional text. However, if it is set to `XmEDIT_LOGICAL`, the selection may not be contiguous because there is no one-to-one correspondence between the display and the input characters in the buffer.

To deselect a region with the mouse, move the pointer anywhere outside of the selected region and click SELECT.

To move the insertion cursor without changing the selection, move the pointer to the location where you want to begin typing, hold down Control, and click SELECT.

Text components provide four ways to copy or move text within the same component or from one component to another:

- clipboard transfer
- drag transfer
- primary transfer
- quick transfer

The text must be selected before copying or moving, and the selection is influenced by the `XmNeditPolicy` resource.

To Use Clipboard Transfer

1. Select the text that you want to copy or move.
2. Press Control-Insert to copy the selection to the clipboard or Shift-Delete to cut it to the clipboard.
3. Move the insertion cursor to the location where you want to insert the text.
4. Press Shift-Insert to paste the text in the new location.

> During deletion and insertion, if the selection contains bi-directional text, the text inserted may not be the original string.

For a discussion on drag transfer, primary transfer, and quick transfer, see *Motif User's Guide*.

To delete text within a Text component, use the Backspace or Delete keys. If text is selected, either key will delete the selection. If there is no text selected, Backspace deletes the character preceding the cursor if the `XmNlayoutDirection` is left-to-right, and deletes the character following the cursor if the `XmNlayoutDirection` is right-to-left. Similarly, Delete deletes the character following the cursor if the `XmNlayoutDirection` is left-to-right, and deletes the character preceding the cursor if the `XmNlayoutDirection` is right-to-left.

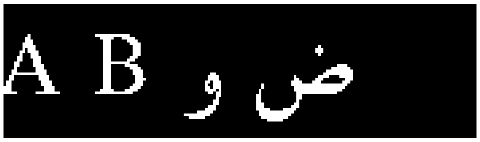
3 Addendum to Motif *Programmer's Guide* (Revision 2.1)

3.1 Layout Direction

The direction of a compound string is stored so that the data structure will be equally useful for describing text in left-to-right languages such as English, Spanish, French, and German, as well as for text in right-to-left languages, such as Hebrew and Arabic. In Motif applications, you can set the layout direction using the `XmNlayoutDirection` resource from the VendorShell or MenuShell. Manager and Primitive widgets (as well as Gadgets) also have an `XmNlayoutDirection` resource. The default value is inherited from the closest ancestor that has the same resource.

In the case of an `XmText` widget, you need to specify the vertical direction as well. Setting the `layoutDirection` to `XmRIGHT_TO_LEFT` will result in the string direction from right-to-left, but the cursor will move vertically down. If the vertical direction is important and top to bottom is desired, be sure to specify `XmRIGHT_TO_LEFT_TOP_TO_BOTTOM`, which specifies that the components are laid out from right-to-left first and then top-to-bottom, and will result in the desired behavior.

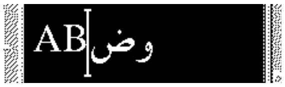
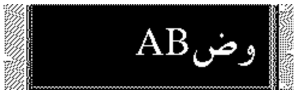
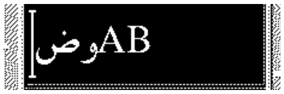
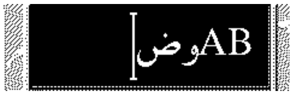
Furthermore, the behavior of `XmText` and `TextField` widgets is influenced by the `XmNalignment` and `XmNlayoutModifier` resources of the `XmRendition`. These resources, in addition to `XmNlayoutDirection`, control the layout behavior of the Text widget. This can be illustrated using the example below.

The input string used in the illustration is 

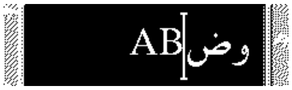
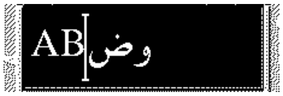
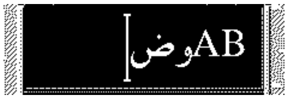
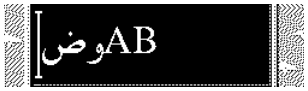
A B ض و

The `XmNlayoutModifier` string `@ls orientation=` setting values for this illustration are shown in the left column.

Layout Direction: XmLEFT_TO_RIGHT

	XmALIGNMENT_BEGINNING	XmALIGNMENT_END
@ls orientation= ltr:ltr		
@ls orientation= rtl:rtl		

Layout Direction: XmRIGHT_TO_LEFT

	XmALIGNMENT_BEGINNING	XmALIGNMENT_END
@ls orientation= ltr:ltr		
@ls orientation= rtl:rtl		

As the illustration shows, `XmNAlignment` dictates whether the text is flush-right or -left in conjunction with the layout direction. On the other hand, `XmNlayoutModifier` breaks the text into segments and arranges them left-to-right or right-to-left depending on the orientation value. In other words, if the `XmNlayoutDirection` is `XmRIGHT_TO_LEFT`, and the `XmNAlignment` value is `XmALIGNMENT_BEGINNING`, the string will be flush-right.

3.2 Creating a Rendition

The following code creates an `XmLabel` whose `XmNlabelString` is of the type `XmCHARSET_TEXT`, using the Rendition whose tag is “ArabicShaped.” The Rendition is created with an `XmNlayoutAttrObject` of “ar“ (corresponding to the locale name for the Arabic locale) and a layout modifier string that specifies for the output buffer a `Numerals` value of `NUMERALS_CONTEXTUAL` and a `ShapeCharset` value of “unicode-1.“

The locale-specific layout module will transform its input text (in this example encoded in ISO 8859-6) in an output buffer of physical characters encoded using the 16-bit Unicode 2.0 codeset. Since an explicit layout locale has been specified, this text will be rendered properly independent of the runtime locale setting.

```
int n;
Arg args[10];
Widget w;
XmString labelString;
XmRendition rendition;
XmStringTag renditionTag;
XmRenderTable renderTable;

      /* alef lam baa noon taa - iso8859-6 */
labelString = XmStringGenerate("\307\344\310\346\312\", NULL
                             XmCHARSET_TEXT, "ArabicShaped");
```

```
w = XtVaCreateManagedWidget("a label", xmLabelWidgetClass, parent,
                             XmNlabelString, labelString,
                            XmNlabelType, XmSTRING,
                             NULL);

n = 0;
XtSetArg(args[n], XmNfontName, "-*-*-medium-r-normal-*-24-*-*-*-*-*-*-*");
      n++;
XtSetArg(args[n], XmNfontType, XmFONT_IS_XOC); n++;
XtSetArg(args[n], XmNlayoutAttrObject, "ar"); n++;
XtSetArg(args[n], XmNlayoutModifier,
          "@ls numerals=:contextual, shapecharset=iso8859-6"); n++;
renditionTag = (XmStringTag) "ArabicShaped";
rendition = XmRenditionCreate(w, renditionTag, args, n);

renderTable =
    XmRenderTableAddRenditions(NULL, &rendition, 1, XmREPLACE_MERGE);

XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
```

3.2.1 Editing a Rendition

The following code creates a TextField widget and a RenderTable with a single Rendition. Note that both the XmNlayoutAttrObject and XmNlayoutModifier pseudo resources have been left unspecified and will therefore default to NULL. This means the LayoutObject associated with the Rendition will be the default locale's, if one exists.

For this example to work properly, the locale must be set to one whose codeset is ISO 8859-6 and whose locale-specific layout module can support the IMPLICIT_BASIC algorithm. It then modifies the Rendition's LayoutObject's ImplicitAlg value via the Rendition's XmNlayoutModifier pseudo resource.

```
int n;
Arg args[10];
Widget w;
 XmRendition rendition;
XmStringTag renditionTag;
XmRenderTable renderTable;

w = XmCreateTextField(parent, "text field", args, 0);

n = 0;
 XtSetArg(args[n], XmNfontName, "-*-*-medium-r-normal-*-24-*-*-*-*-*-*-*");
     n++;
 XtSetArg(args[n], XmNfontType, XmFONT_IS_XOC); n++;
renditionTag = (XmStringTag) "ArabicShaped";
rendition = XmRenditionCreate(w, renditionTag, args, n);

renderTable =
    XmRenderTableAddRenditions(NULL, &rendition, 1, XmREPLACE_MERGE);

XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
```

```
 ....

n = 0;
XtSetArg(args[n], XmNlayoutModifier, "@ls implicitalg=basic");
     n++;
XmRenditionUpdate(rendition, args, n);
```

3.2.2 Related Information

See also XmDirection, XmText.

3.3 Creating a Render Table in a Resource File

Renditions and render tables may be specified in resource files. For properly internationalized application, in fact, this is the preferred method. When the render tables are specified in a file, the program binaries are made independent of the particular needs of a given locale, and may be easily customized to local needs.

Render tables are specified in resource files with the following syntax: *resource_spec*:[*tag*[,*tag*]*]

where tag is some string suitable for the XmNtag resource of a rendition.

This line creates an initial render table containing one or more renditions as specified. The renditions are attached to the specified tags

resource_spec[*|.] *rendition*[*|.]*resource_name*:*value*

The following examples illustrate the CTL resources related to XmRendition that can be set using resource files. The fontType must be set to FONT_IS_XOC for the layout object to take effect. The layoutModifier specified using @ls is passed on to the layout object by the rendition object.

For a complete list of resources that can be set on the layout object using layoutModifier, see *CAE Specification: Portable Layout Services: Context-dependent and Directional Text,* The Open Group: Feb 1997; ISBN 1-85912-142-X; document number C616.

```
*List.renderTable: variable
*List.renderTable.variable.fontType: FONT_IS_XOC
*List.renderTable.variable.layoutAttrObject: ar
*List.renderTable.variable.layoutModifier: @ls numerals=nominal:national, orientation=rtl:rtl
*List.renderTable.variable.fontName: -*-*-medium-r-normal-*-24-*-*-*-*-*-*
```

3.4 Creating a Render Table in an Application

Before creating a render table, an application program must first have created at least one of the renditions that will be part of the table. The XmRenderTableAddRenditions function, as its name implies, is also used to augment a render table with new renditions. To create a new render table, call the XmRenderTableAddRenditions function with a NULL argument in place of an existing render table.

The following code creates a render table using a rendition created with XmNfontType set to XmFONT_IS_XOC.

```
int n;
Arg args[10];
```

```
Widget w;
XmString labelString;
XmRendition rendition;
XmStringTag renditionTag;
XmRenderTable renderTable;

      /* alef lam baa noon taa - iso8859-6 */
labelString = XmStringGenerate("\307\344\310\346\312\", NULL
                              XmCHARSET_TEXT, "ArabicShaped");

w = XtVaCreateManagedWidget("a label", xmLabelWidgetClass, parent,
                            XmNlabelString, labelString,
                           XmNlabelType, XmSTRING,
                            NULL);

n = 0;
XtSetArg(args[n], XmNfontName, "-*-*-medium-r-normal-*-24-*-*-*-*-*-*-*");
     n++;
XtSetArg(args[n], XmNfontType, XmFONT_IS_XOC); n++;
XtSetArg(args[n], XmNlayoutAttrObject, "ar"); n++;
XtSetArg(args[n], XmNlayoutModifier,
          "@ls numerals=nominal:contextual, shapecharset=iso8859-6"); n++;
renditionTag = (XmStringTag) "ArabicShaped";
rendition = XmRenditionCreate(w, renditionTag, args, n);

renderTable =
    XmRenderTableAddRenditions(NULL, &rendition, 1, XmREPLACE);

XtVaSetValues(w, XmNrenderTable, renderTable, NULL);
```

3.5 Horizontal Tabs

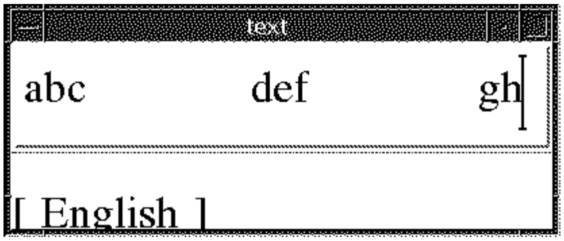
To control the placement of text, a compound string can contain tab characters. To interpret those characters on display, a widget will refer to the rendition in effect for that compound string, where it will find a list of tab stops. However, the dynamic widgets (`TextField` and `XmText`) do not use the tab resource of the rendition. Instead, they compute the tab width using the formula of `8*(width of character 0)`.

The tab measurement is the distance from the left margin of the compound string display, or from the right margin if the layout direction is right-to-left. It is important to note that regardless of the directionality of the text (Arabic right-to-left or English left-to-right) the tab will insert space to the right or left as specified by the layout direction (`XmNlayoutDirection`).

The text following a tab is always aligned at the tab stop, and the tab stop is calculated from the start of the widget, which in turn is influenced by `XmNlayoutDirection`. The behavior of the tabs and their interaction with directionality of the text and the `XmNlayoutDirection` of the widget is illustrated in Figure 3–1, *Tabbing Behavior*.

The input for this illustration is `abc\tdef\tgh`.

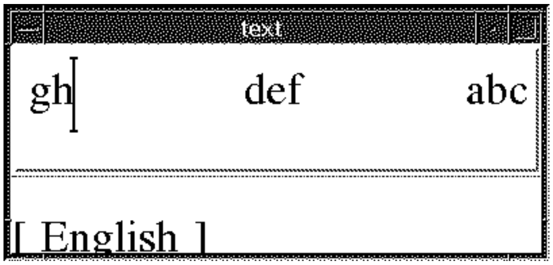
Figure 3–1 Tabbing Behavior



abc def gh

[English]

Layout Direction: `XmLEFT_TO_RIGHT`



gh def abc

[English]

Layout Direction: `XmRIGHT_TO_LEFT`

3.6 Mouse Selection

The user makes a primary selection with SELECT (the left mouse button). Pressing SELECT deselects any existing selection and moves the insertion cursor and the anchor to the position in the text where the button is pressed. Dragging SELECT selects all text between the anchor and the pointer position, deselecting any text outside the range.

The text selected is influenced by the resource `XmNeditPolicy`, which can be set to `XmEDIT_LOGICAL` or `XmEDIT_VISUAL`. If the `XmNeditPolicy` is set to `XmEDIT_LOGICAL`, and if the text selected is bi-directional in nature, the selected text will not be contiguous visually and will be a collection of segments. This is because the text in the logical buffer does not have a one-to-one correspondence with the display.

As a result, the contiguous buffer of logical characters of bi-directional text when rendered will not result in a continuous stream of characters. Conversely, when the `XmNeditPolicy` is set to `XmEDIT_VISUAL`, the text selected may be contiguous visually but will be segmented in the logical buffer. So the sequence of selection, deletion, and insertion of bi-directional text at the same cursor point will not result in the same string.

3.7 Keyboard Selection

The selection operation available with the mouse is also available with keyboard. The combination of Shift-arrow keys will allow the selection of text.

The text selected is influenced by the resource `XmNeditPolicy`, which can be set to `XmEDIT_LOGICAL` or `XmEDIT_VISUAL`. If the `XmNeditPolicy` is set to `XmEDIT_LOGICAL`, and if the text selected is bi-directional in nature, the selected text will not be contiguous visually and will be a collection of segments. This is because the text in the logical buffer does not have one to one correspondence with the display, as a result, the contiguous buffer of logical characters of bi-directional text when rendered will not result in a continuous stream of characters.

Conversely, when the `XmNeditPolicy` is set to `XmEDIT_VISUAL`, the text selected may be contiguous visually but will be segmented in the logical buffer. So the sequence of selection, deletion, and insertion of bi-directional text at the same cursor point will not result in the same string.

3.8 Text Resources and Geometry

Text has several resources that relate to geometry, including the following:

- The render table `XmNrenderTable` that the widget uses to select a font or font set and other attributes in which to display the text.

  The `Text` and `Textfield` widgets can use only the font-related rendition resources, such as `XmNfontType`, and can also specify the attributes of the layout object, such as `XmNlayoutAttrObject`, usually a locale identifier, and `XmNlayoutModifier`, which specifies the layout values to be passed through to the Layout Object associated with the XOC associated with this `XmRendition`.

- A resource (`XmNwordWrap`) that specifies whether lines are broken at word bounderies when the text would be wider than the widget.

  Breaking a line at a word boundary does not insert a new line into the text. In the case of cursive languages like Arabic, if the word length is greater than the widget length, the word is wrapped to the next line, but the first character in the next line is shaped independently of the previous character in the logical buffer.

4 Addendum to Motif *Style Guide* (Revision 2.1)

4.1 Text Cursor

4.1.1 Description

A `Text` component should be used to display and enter text, and must be composed of an area for displaying and entering text. The text can be either a single line or multiple lines. Text must support bi-directional text and vertical text. In addition, the text should support both the Visual and Logical edit policies. Selection, deletion, and cursor movement are affected by the edit policy as well as the directionality of the text.

4.1.2 Navigation

- ←

  Must move the location cursor left one character. If the `XmNeditPolicy` is `XmEDIT_LOGICAL` and the directionality of the text is right-to-left, the cursor will move to the beginning of the next logical character on the left.

- →

  Must move the location cursor right one character. If the edit policy is logical and the directionality of the text is right-to-left, the cursor will move to the beginning of the next logical character on the right.

- Control →

  In a Text component used generally to hold multiple words, must move the location cursor to the right by one word. That is, Control → must place the location cursor before the first character that is not a space, tab, or newline character. However, if the edit policy is logical and the directionality of the text is right-to-left, the cursor will move to the beginning of the next logical word on the right.

- Control ←

  In a Text component used generally to hold multiple words, must move the location cursor to the left by one word. That is, Control ← must place the location cursor before the first character that is not a space, tab, or newline character. However, if the edit policy is logical and the directionality of the text is right-to-left, the cursor will move to the beginning of the next logical word on the left.

4.2 Displaying Text

The CTL extension (Xm CTL) allows Motif to support languages like Arabic, Hebrew, and Thai, whose script and writing systems require complex transformations between the logical and physical representations of text. Specifically, Xm CTL supports character shaping (positional variant selection, ligation, diacritics, symmetrical swapping, and national numerals) and segment reordering.

The only impact Xm CTL has on the Motif API is the introduction of a few new resources and resource values. Its primary public interface changes are to widget behaviors and depend on the locale in use, the runtime values of CTL-sensitive resources, and of course, the text content that requires layout services.

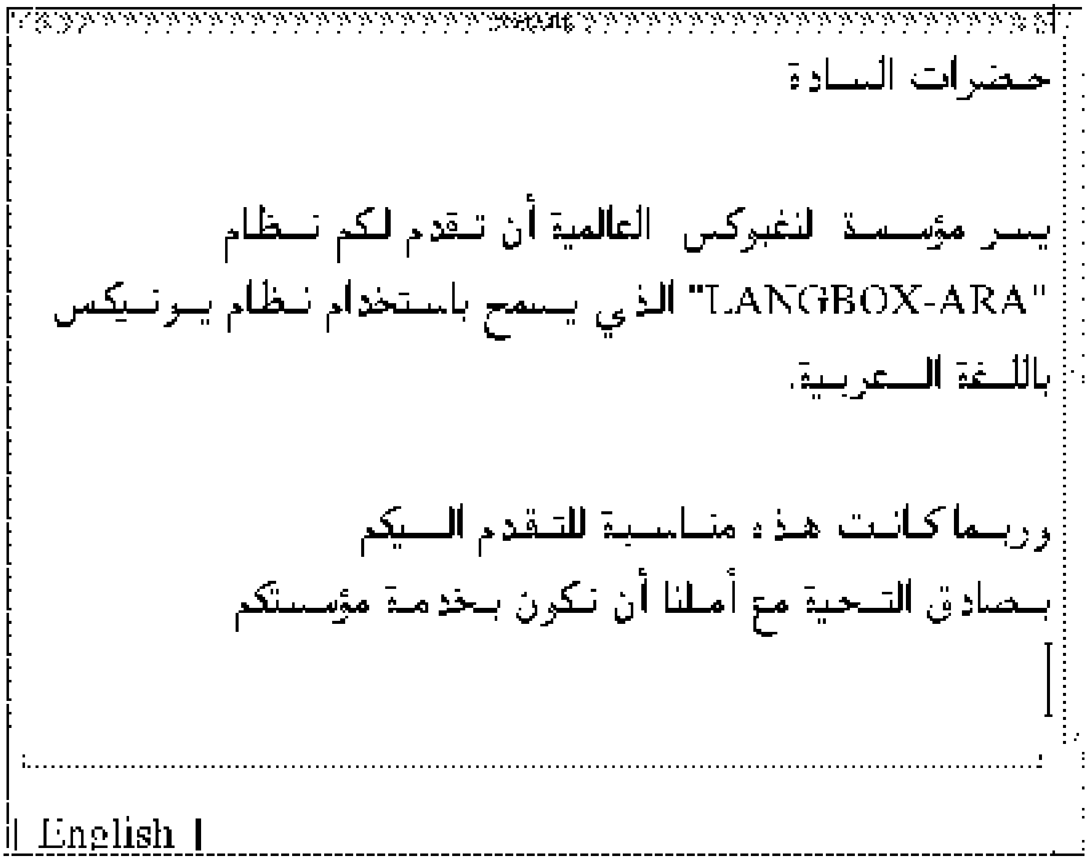
For static text (`XmString`, `XmLabel`, `XmLabelGadget`, and `XmList`), all of the Xm CTL functionality is added via the Motif 2.0+ Rendition/RenderTable abstraction. For dynamic text (`XmText` and `XmTextField`), Xm CTL requires some further enhancement of the widgets themselves. The new dynamic text behaviors that will be most visible to the user are those dealing with bi-directional text.

Specifically, Xm CTL supports the following complex-language shaping and reordering features provided by underlying locale-dependent PLS[1] module transformations:

- positional variation
- ligation (many-to-one) and character composition (one-to-many)
- diacritics
- bi-directionality
- symmetrical swapping
- numeral shaping
- string validation

The following illustrates these features.

[1] Xm CTL is built on top of the X/Open Portable Layout Service (“PLS“) specification. For more information, see *CAE Specification: Portable Layout Services: Context-dependent and Directional Text*.



حضرات السادة

يسر مؤسسة لنغبوكس العالمية أن تقدم لكم نظام
"LANGBOX-ARA" الذي يسمح باستخدام نظام يونيكس
باللغة العربية.

وربما كانت هذه مناسبة للتقدم اليكم
بصادق التحية مع أملنا أن نكون بخدمة مؤسستكم

4.2.1 Static Text Overview

For static text, the only relevant Xm CTL extensions are those pertaining to `XmDirection` and `XmRendition`. By setting the `XmNlayoutAttrObject` and `XmNlayoutModifier` pseudo resources, the `LayoutObject` can be configured as needed. Note that it is not a requirement that all text components of a given `XmString` be in the same locale, since different Rendition's in the RenderTable can have different locales and charsets, and therefore different `LayoutObject`s.

Sharing `LayoutObject`s is achieved by sharing the RenderTable or Rendition, as is also the case with the CDE/Motif 1 `XmFontList` or `XmFontListEntry` objects (now deprecated). There is no Shell-level cache of `LayoutObject`s to be shared among descendant widgets in the same way as input method sharing using the `XmNinputPolicy` resource.

4.2.2 Dynamic Text Overview

The dynamic text widgets `XmText` and `XmTextField` also rely on extensions to `XmDirection` and `XmRendition` alluded to in the previous section. In addition, control over the visual or logical “editing mode“ of the widget is provided by a new set of routines that enable visual mode, and a new resource. (CDE/Motif.next is by default logical in its documented behavior.)

As described in the previous section, the dynamic widgets depend on the configuration of the `LayoutObject` to produce meaningful behavior. As far as the public interfaces to CDE/Motif.next are concerned, the layout transformation output buffer will be seen on the screen, and the input buffer will be the content of the widget for all other purposes, including the programmatic setting of selections or cursor/insertion positions.

4.3 Definitions and Examples

This section contains definitions of new terms and examples of text insertion.

4.3.1 Definitions

Insertion Point

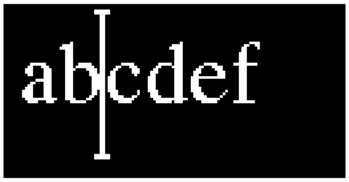
> The location where typed text is inserted. Note that this refers to the position in the input buffer.

Cursor Position

> The location where the cursor is displayed. The cursor position always indicates the insertion point. Note that this is the pixel position on the screen.

4.3.2 Text Insertion Examples

In the following example of the English text “abcdef,” the cursor position in the input buffer is “Insertion point : 3.” The cursor position is between “b” and “c,” and can be seen as ending the pixel position of “b” or starting pixel of “c;” both are the same.

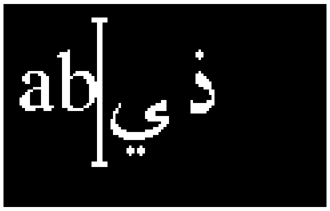


abcdef

Motif has two options available when drawing the cursor:

- Draw the cursor at the ending pixel of the character “b,” or
- Draw the cursor at the starting pixel of the character “c”

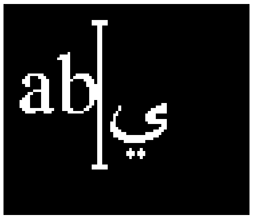
For unidirectional text like plain English or plain Arabic, the cursor positions for both coincide, whereas for bi-directional text they need not. The following example makes this distinction clear. The lowercase characters are English and the uppercase characters are Arabic (abCD), which are shown next to the equivalent English characters. The cursor position in the input buffer is “Insertion point : 3” because of the right to left characteristics of the segment “AB.”



abذي

Note that the pixel position after “b” is no longer the same as the pixel position before “C.” If Motif displays the cursor at the end of the English character (after “b”), then the following behavior would occur.

If the user pressed the Delete key with the cursor following “b,” the result would be:



Input Buffer:abD

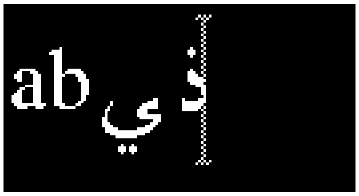
Screen Output : abD

Insertion Point: 3

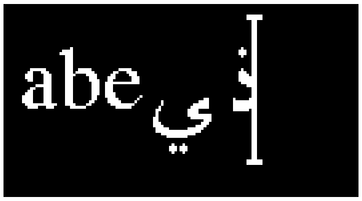
Cursor Position: after “b”

In other words, the deleted character is not visually nearest the cursor.

The cursor at the starting pixel position of “C” would appear as follows:



If the user typed an English character (for example, “e”), the text would display as follows:



In other words, the typed character is inserted far away from the cursor, which appears non-intuitive.

The problems described in this section could be resolved if dual or split cursors were implemented. This solution may be examined for the next version.