Xm Data Types

XmDirection

XmFontList

XmParseMapping

XmParseTable

XmRenderTable

XmString

XmStringDirection

XmStringTable

XmTab

XmTabList

XmTextPosition

XmDirection (library call)

NAME

XmDirection –Data type for the direction of widget components

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmDirection is the data type specifying the direction in which the system displays subwidgets, children of widgets, or other visual components that are to be laid out. This data type also affects traversal order within tab groups.

XmDirection is implemented as an unsigned char bit mask. The horizontal and vertical directions can be specified independent of each other. **XmDirection** also specifies the precedence of the horizontal and vertical directions relative to each other. For example, a value of **XmRIGHT_TO_LEFT_TOP_TO_BOTTOM** lays out a component horizontally from right to left first, then vertically top to bottom.

XmDirection provides the following masks, each of which corresponds to a particular bit in **XmDirection**:

- **XmRIGHT_TO_LEFT_MASK**
- **XmLEFT_TO_RIGHT_MASK**
- **XmTOP_TO_BOTTOM_MASK**
- **XmBOTTOM_TO_TOP_MASK**
- **XmPRECEDENCE_HORIZ_MASK**
- **XmPRECEDENCE_VERT_MASK**

In addition to the preceding single bit masks, **XmDirection** also provides the following multiple bit masks. These multiple bit masks are particularly useful as arguments to **XmDirectionMatchPartial**:

- **XmHORIZONTAL_MASK**
- **XmPRECEDENCE_MASK**
- **XmVERTICAL_MASK**

Motif also provides the following enumerated constants for specifying various combinations of directions:

XmRIGHT_TO_LEFT_TOP_TO_BOTTOM

Specifies that the components are laid out from right to left first, then top to bottom.

XmLEFT_TO_RIGHT_TOP_TO_BOTTOM

Specifies that the components are laid out from left to right first, then top to bottom.

XmRIGHT_TO_LEFT_BOTTOM_TO_TOP

Specifies that the components are laid out from right to left first, then bottom to top.

XmLEFT_TO_RIGHT_BOTTOM_TO_TOP

Specifies that the components are laid out from left to right first, then bottom to top.

XmTOP_TO_BOTTOM_RIGHT_TO_LEFT

Specifies that the components are laid out from top to bottom first, then right to left.

XmTOP_TO_BOTTOM_LEFT_TO_RIGHT

Specifies that the components are laid out from top to bottom first, then left to right.

XmBOTTOM_TO_TOP_RIGHT_TO_LEFT

Specifies that the components are laid out from bottom to top first, then right to left.

XmBOTTOM_TO_TOP_LEFT_TO_RIGHT

Specifies that the components are laid out from bottom to top first, then left to right.

XmTOP_TO_BOTTOM

Specifies that the components are laid out from top to bottom. If horizontal direction is important, do not use this constant.

XmBOTTOM_TO_TOP

Specifies that the components are laid out from bottom to top. If horizontal direction is important, do not use this constant.

XmDEFAULT_DIRECTION

Specifies that the components are laid out according to the default direction. (This constant is primarily for widget writers.)

XmLEFT_TO_RIGHT

Specifies that the components are laid out from left to right. If vertical direction is important, do not use this constant.

XmRIGHT_TO_LEFT

Specifies that the components are laid out from right to left. If vertical direction is important, do not use this constant.

RELATED

XmDirectionMatch(3) , **XmDirectionMatchPartial**(3) , **XmDirectionToStringDirection**(3) , **XmString**(3) , **XmStringDirection**(3) , and **XmStringDirectionToDirection**(3) .

XmFontList (library call)

NAME

XmFontList - Data type for a font list

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmFontList is the data type for a font list. A font list consists of font list entries. Each entry contains a font or a font set (a group of fonts) and is identified with a tag, which is optional. If this tag is NULL, the tag is set to **XmFONTLIST_DEFAULT_TAG**.

The value of **XmFONTLIST_DEFAULT_TAG** is **XmFONTLIST_DEFAULT_TAG_STRING**.

When a compound string is displayed, the font list element tag of the compound string segment is matched with a font list entry tag in the font list and the matching font list entry is used to display the compound string. A font list entry is chosen as follows:

- The first font list entry whose tag matches the tag of the compound string segment is used.
- If no match has been found and if the tag of the compound string segment is **XmFONTLIST_DEFAULT_TAG**, the first font list entry whose tag matches the tag that would result from creating an entry with **XmSTRING_DEFAULT_CHARSET** is used. For example, if creating an entry with **XmSTRING_DEFAULT_CHARSET** would result in the tag *ISO8859−1*, the compound string segment tag **XmFONTLIST_DEFAULT_TAG** matches the font list entry tag *ISO8859−1*.
- If no match has been found and if the tag of the compound string segment matches the tag that would result from creating a segment with **XmSTRING_DEFAULT_CHARSET**, the first font list entry whose tag is **XmFONTLIST_DEFAULT_TAG** is used.
- If no match has been found, the first entry in the font list is used.

The font list interface consists of the routines listed in *Related Information*.

Font lists are specified in resource files with the following syntax:
resource_spec: *font_entry* [, *font_entry*]+

The resource value string consists of one or more font list entries separated by commas. Each *font_entry* identifies a font or font set and an optional font list entry tag. A tag specified for a single font follows the font name and is separated by = (equals sign); otherwise, in a font set the tag is separated by a colon. The colon is required whether a tag is specified or not. A font entry uses the following syntax to specify a single font:
font_name ['=' *tag*]

For example, the following entry specifies a 10 point Times Italic font without a font list entry tag;

```
*fontList:  -Adobe-Times-Medium-I-Normal--10*
```

A font entry containing a font set is similar, except a semicolon separates multiple font names and the specification ends with a colon followed by an optional tag:
font_name [';' *font_name*]+ ':' [*tag*]

A *font_name* is an X Logical Font Description (XLFD) string and *tag* is any set of characters from ISO646IRV except space, comma, colon, equal sign and semicolon. Following is an example of a font set entry. It consists of three fonts (except for charsets), and an explicit font list entry tag.

```
*fontList: -Adobe-Courier-Bold-R-Normal--25-180-100-100-M-150;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-240;\
-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-120:MY_TAG
```

Note that the **XmRenderTable** is another data type that can be used for font lists. Refer to the **XmRenderTable(3)** for details.

RELATED

XmFontListAdd(3), **XmFontListAppendEntry**(3), **XmFontListCopy**(3), **XmFontListCreate**(3), **XmFontListEntryCreate**(3), **XmFontListEntryFree**(3), **XmFontListEntryGetFont**(3), **XmFontListEntryGetTag**(3), **XmFontListEntryLoad**(3),

XmFontListFree(3) , **XmFontListFreeFontContext**(3) , **XmFontListGetNextFont**(3) , **XmFontListInitFontContext**(3) , **XmFontListNextEntry**(3) , **XmFontListRemoveEntry**(3) , **XmRenderTable**(3) , and **XmString**(3) .

XmParseMapping (library call)

NAME

XmParseMapping –Data type for a compound string parse mapping

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmParseMapping is an opaque data type for a parse mapping used by **XmStringParseText** to create a compound string. A parse mapping contains a pattern to be matched in text being parsed to create a compound string. It also contains a compound string, or a function to be invoked to provide a compound string, to be included in the compound string being created whenever the pattern is matched.

An application uses a resource−style interface to specify components for an **XmParseMapping**. **XmParseMappingCreate** creates a parse mapping, using a resource−style argument list. **XmParseMappingGetValues** and **XmParseMappingSetValues** retrieve and set the components of a parse mapping. **XmParseMappingFree** recovers memory used by a parse mapping. **XmParseTable** is an array of **XmParseMapping** objects.

The **XmNinvokeParseProc** resource is a function of type **XmParseProc**, which is defined as follows:
XmIncludeStatus (*XmParseProc) (*text_in_out, text_end, type, tag, entry, pattern_length, str_include, call_data*)
XtPointer **text_in_out*;
XtPointer *text_end*;
XmTextType *type*;
XmStringTag *tag*;
XmParseMapping *entry*;
int *pattern_length*;
XmString **str_include*;
XtPointer *call_data*;

A parse procedure provides an escape mechanism for arbitrarily complex parsing. This procedure is invoked when a pattern in the input text is matched with a pattern in a parse mapping whose **XmNincludeStatus** is **XmINVOKE**.

The input text is a pointer to the first byte of the pattern that was matched to trigger the call to the parse procedure. The parse procedure consumes as many bytes of the input string as it needs and sets the input text pointer to the following byte. It returns a compound string to be included in the compound string being constructed, and it also returns an **XmIncludeStatus** indicating how the returned compound string should be handled. If the parse procedure does not set the input text pointer ahead by at least one byte, the parsing routine continues trying to match the input text with the patterns in the remaining parse mappings in the parse table. Otherwise, the parsing routine begins with the new input text pointer and tries to match the input text with patterns in the parse mappings starting at the beginning of the parse table.

text_in_out

Specifies the text being parsed. The value is a pointer to the first byte of text matching the pattern that triggered the call to the parse procedure. When the parse procedure returns, this argument is set to the position in the text where parsing should resume−−that is, to the byte following the last character parsed by the parse procedure.

text_end

Specifies a pointer to the end of the *text_in_out* string. If *text_end* is NULL, the string is scanned until a NULL character is found. Otherwise, the string is scanned up to but not including the character whose address is *text_end.*

type

Specifies the type of text and the tag type. If a locale tag should be created, *type* has a value of either **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**. If a charset should be created, *type* has a value of **XmCHARSET_TEXT**.

tag

Specifies the tag to be used in creating the result. The type of string tag created (charset or locale) depends on the text type and the passed in *tag* value. If the *tag* value is NULL and if *type* indicates that a charset string tag should be created, the string tag has the value that is the result of mapping **XmSTRING_DEFAULT_CHARSET**. If *type* indicates a locale string tag, the string tag has the value **_MOTIF_DEFAULT_LOCALE**.

entry

Specifies the parse mapping that triggered the call to the parse procedure.

pattern_length

Specifies the number of bytes in the input text, following *text_in_out*, that constitute the matched pattern.

str_include

Specifies a pointer to a compound string. The parse procedure creates a compound string to be included in the compound string being constructed. The parse procedure then returns the compound string in this argument.

call_data

Specifies data passed by the application to the parsing routine.

The parse procedure returns an **XmIncludeStatus** indicating how *str_include* is to be included in the compound string being constructed. Following are the possible values:

XmINSERT

Concatenate the result to the compound string being constructed and continue parsing.

XmTERMINATE

Concatenate the result to the compound string being constructed and terminate parsing.

New Resources

The following table defines a set of resources used by the programmer to specify data. The codes in the access column indicate if the given resource can be set at creation time (C), set by using **XmParseMappingSetValues** (S), retrieved by using **XmParseMappingGetValues** (G), or is not applicable (N/A).

XmParseMapping Resource Set				
Name	*Class*	*Type*	*Default*	*Access*
XmNclientData		XtPointer	NULL	CSG
XmNincludeStatus		XmIncludeStatus	XmINSERT	CSG
XmNinvokeParseProc		XmParseProc	NULL	CSG
XmNpattern		XtPointer	NULL	CSG
XmNpatternType		XmTextType	XmCHARSET_TEXT	CSG
XmNsubstitute		XmString	NULL	CSG

XmNclientData

Specifies data to be used by the parse procedure.

XmNincludeStatus

Specifies how the result of the mapping is to be included in the compound string being constructed. Unless the value is **XmINVOKE**, the result of the mapping is the value of **XmNsubstitute**. Following are the possible values for **XmNincludeStatus**:

XmINSERT

Concatenate the result to the compound string being constructed and continue parsing.

XmINVOKE

Invoke the **XmNinvokeParseProc** on the text being parsed and use the returned compound string instead of **XmNsubstitute** as the result to be inserted into the compound string being constructed. The include status returned by the parse procedure (**XmINSERT** or **XmTERMINATE**) determines how the returned compound string is included.

XmTERMINATE

Concatenate the result to the compound string being constructed and terminate parsing.

XmNinvokeParseProc

Specifies the parse procedure to be invoked when **XmNincludeStatus** is **XmINVOKE**.

XmNpattern

Specifies a pattern to be matched in the text being parsed. This is a maximum of one character.

XmNpatternType

Specifies the type of the pattern that is the value of **XmNpattern**. Following are the possible values:

XmCHARSET_TEXT
XmMULTIBYTE_TEXT
XmWIDECHAR_TEXT

XmNsubstitute

Specifies the compound string to be included in the compound string being constructed when **XmNincludeStatus** is **XmINSERT** or **XmTERMINATE**.

RELATED

XmParseMappingCreate(3) , **XmParseMappingFree**(3) , **XmParseMappingGetValues**(3) , **XmParseMappingSetValues**(3) , **XmParseTable**(3) , and **XmString**(3) .

XmParseTable (library call)

NAME

XmParseTable – Data type for a compound string parse table

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmParseTable is the data type for an array of parse mappings (objects of type **XmParseMapping**).

A parse table is used by some routines that parse and unparse compound strings. The table is an ordered list of parse mappings. A parsing routine that uses a parse table scans the input text and searches the parse mappings, in order, for one containing a pattern that matches the input text. The matching parse mapping supplies a compound string to be included in the compound string under construction.

An unparsing routine that uses a parse table searches the parse mappings, in order, for one containing a compound string that matches the input compound string. The unparsing routine can then include the parse mapping's text pattern in the output text under construction.

RELATED

XmParseMapping(3) , **XmParseTableFree**(3) , and **XmString**(3) .

XmRenderTable (library call)

NAME

XmRenderTable – Data type for a render table

SYNOPSIS

#include <Xm/Xm.h>
XmRenderTable

DESCRIPTION

XmRenderTable is the data type for a render table, which contains a table of renditions (**XmRendition**s). Each rendition consists of a set of attributes for rendering text, including a font or fontset, colors, tabs, and lines. Each rendition is identified with a tag.

When a compound string is displayed, for each segment in the string, the rendition used to render that string is formed as follows. If the segment has at least one rendition begin tag or if the list of tags formed by accumulating from previous segments the rendition begin tags and removing the rendition end tags is not empty, these tags are matched with renditions in the render table. The effective rendition used to render the segment is formed by successively merging each rendition found into the current rendition with non–**XmAS_IS** (**XmUNSPECIFIED_PIXEL** for color resources) values for resources in the rendition to be merged, thus replacing the corresponding values of the resources in the current rendition. Finally, if the resulting rendition still has resources with unspecified values and the segment has a locale or charset tag (these are optional and mutually exclusive) this tag is matched with a rendition in the render table, and the missing rendition values are filled in from that entry.

If no matching rendition is found for a particular tag, then the **XmNnoRenditionCallback** of the **XmDisplay** object is called and the render table is searched again for that tag.

If the resulting rendition does not specify a font or fontset, then for segments whose text type is **XmCHARSET_TEXT**, the render table will be searched for a rendition tagged with **XmFONTLIST_DEFAULT_TAG**, and if a matching rendition is found, it will be merged into the current rendition. If the resulting rendition contains no font or fontset, the **XmNnoFontCallback** will be called with the default rendition and "" as the font name. If no rendition matches or no font was found after the callback, then the first rendition in the render table will be merged into the current rendition. If this rendition still has no font, then the segment will not be rendered and a warning will be emitted.

For segments whose text type is **XmMULTIBYTE_TEXT** or **XmWIDECHAR_TEXT**, the render table will be searched for a rendition tagged with **_MOTIF_DEFAULT_LOCALE**, and, if a matching rendition is found, it will be merged into the current rendition. If the resulting rendition contains no font, the **XmNnoFontCallback** will be called with the default rendition and "" as the font name. An application can have this callback attempt to remedy this problem by calling **XmRenditionUpdate** on the input rendition to provide a font for the widget to use. This may be done by either providing an alternative font name to be loaded using the **XmNfontName** and **XmNfontType** resources or with an already loaded font using the **XmNfont** resource. If no rendition matches or no font was found after the callback, then the segment will not be rendered and a warning will be issued.

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.

If no tags are specified, then a render table will be created that is either empty or contains only a rendition with a tag of **_MOTIF_DEFAULT_LOCALE**.

Specific values for specific rendition resources are specified using the following syntax:
resource_spec [*|.] *rendition*[*|.] *resource_name*: *value*

where:

resource_spec

Specifies the render table.

rendition

Is either the class Rendition or a tag.

resource_name

Is either the class or name of a particular resource.

value

Is the specification of the value to be set.

Any resource line that consists of just a resource name or class component with no rendition component or loose binding will be assumed to specify resource values for a rendition with a tag of **_MOTIF_DEFAULT_LOCALE**. In effect, this creates a default rendition in much the same way that specifying no fontlist tag for a fontlist resource causes the fontlist created to contain an entry tagged with **XmFONTLIST_DEFAULT_TAG**:
resource_spec.resource_name: *value*

For example, the following would set the **XmNrenderTable** resource of *label1* to a render table consisting of three renditions tagged with **_MOTIF_DEFAULT_LOCALE**, *bold*, and *oblique*, with values for resources set as described in the resource specifications.

```
*label1.renderTable: bold, oblique
*label1.renderTable.bold.renditionForeground: Green
*label1.renderTable.bold.fontName: *-*-*-bold-*-iso8859-1
*label1.renderTable.bold.fontType: FONT_IS_FONT
*label1.renderTable.oblique.renditionBackground: Red
*label1.renderTable.oblique.fontName: *-*-*-italic-*-iso8859-1
*label1.renderTable.oblique.fontType: FONT_IS_FONT
*label1.renderTable.oblique.underlineType: AS_IS
*label1.renderTable.fontName: fixed
*label1.renderTable.fontType: FONT_IS_FONT
*label1.renderTable.renditionForegound: black
*label1.renderTable*tabList: 1in, +1.5in, +3in
```

RELATED

XmRenderTableAddRenditions(3) , **XmRenderTableCopy**(3) , **XmRenderTableCvtFromProp**(3) , **XmRenderTableCvtToProp**(3) , **XmRenderTableFree**(3) , **XmRenderTableGetRendition**(3) , **XmRenderTableGetRenditions**(3) , **XmRenderTableGetTags**(3) , **XmRenderTableRemoveRenditions**(3) , **XmRendition**(3) , and **XmString**(3) .

XmString (library call)

NAME

XmString – Data type for a compound string

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmString is the data type for a compound string. Compound strings consist of a sequence of components, including, but not limited to, the following:

- **XmSTRING_COMPONENT_SEPARATOR**
- **XmSTRING_COMPONENT_TAB**
- **XmSTRING_COMPONENT_LAYOUT_POP**
- **XmSTRING_COMPONENT_DIRECTION**
- **XmSTRING_COMPONENT_LAYOUT_PUSH**
- **XmSTRING_COMPONENT_CHARSET**
- **XmSTRING_COMPONENT_FONTLIST_ELEMENT_TAG**
- **XmSTRING_COMPONENT_LOCALE**
- **XmSTRING_COMPONENT_LOCALE_TEXT**
- **XmSTRING_COMPONENT_TAG**
- **XmSTRING_COMPONENT_TEXT**
- **XmSTRING_COMPONENT_END**
- **XmSTRING_COMPONENT_RENDITION_BEGIN**
- **XmSTRING_COMPONENT_RENDITION_END**
- **XmSTRING_COMPONENT_UNKNOWN**
- **XmSTRING_COMPONENT_WIDECHAR_TEXT**

and also a rendition tags table, text, and text component. When a compound string is displayed, the rendition tags and the direction are used to determine how to display the text.

Calling **XtGetValues** for a resource whose type is **XmString** yields a copy of the compound string resource value. The application is responsible for using **XmStringFree** to free the memory allocated for the copy.

Please see the **XmStringComponentType** reference page for more detail about compound string components, and for a description of the order in which the components should appear in a compound string. Refer to the **XmRenderTable** reference page for a description of the algorithm that associates the rendition tags used for displaying a compound string text component with a rendition in a render table.

RELATED

XmParseMapping(3) , **XmParseMappingCreate**(3) , **XmParseMappingFree**(3) , **XmParseMappingGetValues**(3) , **XmParseMappingSetValues**(3) , **XmParseTable**(3) , **XmParseTableFree**(3) , **XmStringBaseline**(3) , **XmStringByteCompare**(3) , **XmStringByteStreamLength**(3) , **XmStringCompare**(3) , **XmStringComponentCreate**(3) , **XmStringComponentType**(3) , **XmStringConcat**(3) , **XmStringConcatAndFree**(3) , **XmStringCopy**(3) , **XmStringCreate**(3) , **XmStringCreateLocalized**(3) , **XmStringCreateLtoR**(3) , **XmStringCreateSimple**(3) , **XmStringDirection**(3) , **XmStringDirectionCreate**(3) , **XmStringDirectionToDirection**(3) , **XmStringDraw**(3) , **XmStringDrawImage**(3) , **XmStringDrawUnderline**(3) , **XmStringEmpty**(3) , **XmStringExtent**(3) , **XmStringFree**(3) , **XmStringFreeContext**(3) , **XmStringGenerate**(3) , **XmStringGetLtoR**(3) , **XmStringGetNextComponent**(3) , **XmStringGetNextSegment**(3) , **XmStringGetNextTriple**(3) , **XmStringHasSubstring**(3) , **XmStringHeight**(3) , **XmStringInitContext**(3) , **XmStringIsVoid**(3) , **XmStringLength**(3) , **XmStringLineCount**(3) ,

XmStringNConcat(3) , **XmStringNCopy**(3) , **XmStringParseText**(3) , **XmStringPeekNextComponent**(3) , **XmStringPeekNextTriple**(3) , **XmStringPutRendition**(3) , **XmStringSegmentCreate**(3) , **XmStringSeparatorCreate**(3) , **XmStringTable**(3) , **XmStringTableParseStringArray**(3) , **XmStringTableProposeTablist**(3) , **XmStringTableToXmString**(3) , **XmStringTableUnparse**(3) , **XmStringToXmStringTable**(3) , **XmStringUnparse**(3) , **XmStringWidth**(3) , **XmCvtXmStringToByteStream**(3) , **XmCvtXmStringToCT**(3) , **XmCvtCTToXmString**(3) , and **XmCvtByteStreamToXmString**(3) .

XmStringDirection (library call)

NAME

XmStringDirection – Data type for the direction of display in a string

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmStringDirection is the data type for specifying the direction in which the system displays characters of a string, or characters of a segment of a compound string. This is an enumeration with three possible values:

XmSTRING_DIRECTION_L_TO_R

Specifies left to right display

XmSTRING_DIRECTION_R_TO_L

Specifies right to left display

XmSTRING_DIRECTION_DEFAULT

Specifies that the display direction will be set by the widget in which the compound string is to be displayed.

RELATED

XmString(3) .

XmStringTable (library call)

NAME

XmStringTable –Data type for an array of compound strings

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmStringTable is the data type for an array of compound strings (objects of type **XmString**).

RELATED

XmString(3) .

XmTab (library call)

NAME

XmTab – Data type for a tab stop

SYNOPSIS

#include <Xm/Xm.h>
XmTab

DESCRIPTION

XmTab is a data structure that specifies a tab stop to be used in rendering an **XmString** containing tab components. An **XmTab** value contains a value, a unit type, an offset model (either **XmABSOLUTE** or **XmRELATIVE**), an alignment model (**XmALIGNMENT_BEGINNING**), and a decimal point character. The resource file syntax for **XmTab** is specified in the **XmTabList** reference page.

RELATED

XmTabCreate(3), **XmTabFree**(3), **XmTabGetValues**(3), **XmTabList**(3), and **XmTabSetValue**(3).

XmTabList (library call)

NAME

XmTabList – Data type for a tab list

SYNOPSIS

#include <Xm/Xm.h>
XmTabList

DESCRIPTION

XmTabList is the data type for a tab list. A tab list consists of tab stop list entries (**XmTab**s). Whenever a tab component is encountered while an **XmString** is being rendered, the origin of the next X draw depends on the next **XmTab**. If a tab stop would cause text to overlap, the x position for the segment is reset to follow immediately after the end of the previous segment.

Tab lists are specified in resource files with the following syntax:
resource_spec: tab WHITESPACE [, WHITESPACE *tab*]*

The resource value string consists of one or more tabs separated by commas. Each *tab* identifies the value of the tab, the unit type, and whether the offset is relative or absolute. For example:
tab := *float* [WHITESPACE *units*]
float := [*sign*] [[DIGIT]*.]DIGIT+
sign := +

where the presence or absence of *sign* indicates, respectively, a relative offset or an absolute offset. Note that negative tab values are not allowed. *units* indicates the unitType to use as described in the **XmConvertUnits** reference page.

For example, the following specifies a tab list consisting of a one inch absolute tab followed by a one inch relative tab:

```
*tabList: 1in, +1in
```

For resources of type, dimension, or position, you can specify units as described in the **XmNunitType** resource of the **XmGadget**, **XmManager**, or **XmPrimitive** reference page.

RELATED

Refer to the *Motif Programmer's Guide* for more information about tabs and tab lists. **XmTabListCopy(3)** , **XmTabListFree**(3) , **XmTabListGetTab**(3) , **XmTabListInsertTabs**(3) , **XmTabListRemoveTabs**(3) , **XmTabListReplacePositions**(3) , and **XmTabListTabCount**(3) .

XmTextPosition (library call)

NAME

XmTextPosition – Data type for a character position within a text string

SYNOPSIS

#include <Xm/Xm.h>

DESCRIPTION

XmTextPosition is an integer data type that holds a character's position within a text string for Text and TextField.

An **XmTextPosition** value conceptually points to the gap between two characters. For example, consider a text string consisting of *N* characters. A value of 0 refers to the position immediately prior to the first character. A value of 1 refers to the position in between the first and second characters. A value of *N* refers to the position immediately following the last character. Therefore, the text string of *N* characters actually contains *N + 1* positions.

RELATED

XmText(3) .