Customizing Motif Applications

Most Motif applications provide built−in ways for you to change various application resources, such as colors and fonts. Customization functions are often found on the Options Menu on the MenuBar. The customization features provided by an application typically are sufficient for most users.

Every Motif component has numerous resources that control its appearance and behavior. Advanced users may wish to modify certain resources of specific components in an application to change the functionality and appearance of the application beyond what is possible using the customization features that are provided. Since the range of possible modifications is virtually endless, you need to be careful not to change an application so much that it no longer functions properly. Check with the documentation for an application before you start changing resource values. Consult the *Motif Programmer's Reference* for information about the resources that are used by different Motif components.

This chapter describes the following features for customizing Motif applications:

- Customizing text display by specifying colors, font lists, tabs, and other text properties
- Using drag and drop resources
- Using user−customizable component resources

Customizing Text Display

Motif permits users to modify a large number of text properties. The color, fonts, and size of text are all accessible through the modification of resources, as well as tab stops and some miscellaneous other properties such as underlining and strike−through. The data structure controlling these properties is called a *rendition* and is introduced in the following section.

> **Note:** While Motif may allow a user to modify many different aspects of text display, there is no guarantee that a particular application program will continue to function properly once such a modification is made. For example, a user may change the color of some text so that it matches the background color perfectly. This user should not then be surprised that the text in question is rendered illegible. When modifying the resources for an application, it is best to proceed with caution, and with the program's documentation at hand.

Renditions

A rendition is simply a collection of all the information necessary to render text. Each segment of text on display (segments can be as long or as short as circumstances warrant) is tagged with a "tag," indicating which of a list of possible renditions is to be used for that text. That list, containing one or more renditions, is called a *render table* and is described in Section 7.1.2.

> **Note:** Some of the Motif widgets used for text input do not use renditions. This means that a user can change the fonts used in these widgets but not the color or tabs or other rendition properties. In these cases, a font list structure is available that is analogous to the render table.

A rendition is neither a widget nor a gadget. However, the style used to specify resources for widgets and gadgets is a simple and familiar way to set data values, and it has been implemented for renditions and render tables as well. To change the foreground color for a rendition, for example, a user must set a "resource" called *renditionForeground* for that rendition, in the same way a user would change the color resource for some widget.

A rendition can have any of the following resources:

tag

> The name of the rendition. This is simply a character string.

fontName

> The name of the font to use. See Section 7.1.1.1 for information about font naming conventions.

fontType

> There are two different kinds of font types, for two different kinds of languages. One sort of language, such as English, Spanish, Greek, or Swedish, requires only one font to spell any word in that language. For these languages, the font type should be **FONT_IS_FONT**. The other kind of language may require more than one font, in a collection called a *font set* to spell any text in that language. Examples of these languages are Japanese, Chinese, and Korean. For these languages, the font type should be **FONT_IS_FONTSET**.

tabList

> This is a list of tab stops, specified in the format described in Section 7.1.1.2.

renditionForeground

> The color of the characters in the text. (See the documentation of your system for a list of the color names available there.)

renditionBackground

> The color of the cell surrounding each character in the text.

underlineType

> The type of underlining. The possible values of this resource are **NO_LINE**, **SINGLE_LINE**, **DOUBLE_LINE**, **SINGLE_DASHED_LINE**, or **DOUBLE_DASHED_LINE**.

strikethruType

The type of line drawn through the text. The possible values of this resource are **NO_LINE**, **SINGLE_LINE**, **DOUBLE_LINE**, **SINGLE_DASHED_LINE**, or **DOUBLE_DASHED_LINE**.

Any of these resources (except *tag*, *tabList*, and *fontType*) can have the value **AS_IS**, which specifies that the resource take its default value, which is usually inherited from the parent widget. If a *tabList* resource is missing, the widget will usually have default tab stops set. Note that the application foreground and background use unspecified pixel colors.

Fonts

A font is typically identified by such characteristics as its family, point size, weight, slant, and character set. Sometimes a single font may be used to display a given language, as in English, Swedish, Italian, Greek, and most other alphabetic languages. However, some languages may require more than one font to display its characters. A font set is a set of one or more fonts used to display a single language. Fonts from the set are loaded according to a list of base font names supplied by the application and the character sets required by the locale.

A *fontName* can be an X Logical Font Description (XLFD) string, or a site−specific name of a font. When specifying fonts, you can provide a detailed description for a given font, or you can use the * (asterisk) wildcard to indicate fields whose values can be selected by the font−loading routines called by an application. The following resource entry specifies a 12−point Courier font that can be of any weight, slant, or set width. The font is to be part of a rendition called *rend1*.

```
*renderTable.rend1.fontName:  −Adobe−Courier−*−*−*−−12−120−100−100−*−*
*renderTable.rend1.fontType:  FONT_IS_FONT
```

The following example specifies a font set for an application running in a Japanese language environment. The font set contains three fonts in a comma−separated list.

```
*renderTable.rend1.fontName:  −JIS−Fixed−Medium−R−Normal−−26−180−100−100−C−240,\
                              −JIS−Fixed−Medium−R−Normal−−26−180−100−100−C−120,\
                              −Adobe−Courier−Bold−R−Normal−−24−240−100−100−M−100
*renderTable.rend1.fontType:  FONT_IS_FONTSET
```

By definition, when an application creates a font set, the character sets used to display the text in the current locale are automatically established. The advantage of supplying a font set is that you do not need to know the character sets required by the locale; the system determines them for you.

Whether you need to specify fonts or font sets for a font resource is application dependent. Internationalized applications can adopt different approaches to setting the language environment and handling text display. Certain applications can require that you explicitly specify a font, including its character set. Consult the documentation for an application if you need to specify any font resources.

Tab Stops

To create a tab stop in a resource file, use the following syntax:

```
resource_spec  : tab [, 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. The tab stop specification uses the following syntax:

```
tab = [ + ]  float [ units ]
```

The tab contains a decimal number (*float*), and an indication of the units to be used. If no units are specified, the default unit is pixels. The presence or absence of a sign indicates, respectively, a relative offset or an absolute offset model. A relative tab stop is measured from the previous tab stop in the list, while an absolute tab stop is measured from the left margin (or the right margin if the layout direction is right−to −left). Note that negative tab values are not permitted and that, if a tab stop would cause two segments of text to be overwritten, the x position of one segment will be moved (the tab will be ignored).

For example, the following line of a resource file sets a tab list for a rendition (*rend1*) in a render table used by a widget called *List*. It specifies a tab list consisting of a 1−inch absolute tab followed by a 1−inch relative tab (equivalent to a 2−inch absolute tab, unless and until a third tab is inserted between the two).

```
*List.renderTable.rend1.tabList: 1in, +1in
```

The recognized units and their abbreviations include the following:

pixels

pix, pixel, pixels

inches

in, inch, inches

centimeter

cm, centimeter, centimeters

millimeters

mm, millimeter, millimeters

points

pt, point, points

font units

fu, font_unit, font_units

Please refer to the *Motif Programmer's Guide* for a definition of font units, and for more information about tab lists.

Render Tables

Render tables are specified in resource files with the following syntax:

resource_spec `: [` tag `[,` tag `]* ]`

where *tag* is some string suitable to be the name of a rendition. This line creates an initial render table containing one more rendition than the number of tags specified. The renditions are attached to the specified tags, with the untagged rendition going with the tag **_MOTIF_DEFAULT_LOCALE**. If no tags are specified, then a render table will be created that 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 a tag (or is omitted), *resource_name* is the name of a particular resource, and *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 will be assumed to specify resource values for a rendition with a tag of **_MOTIF_DEFAULT_LOCALE**.

For example, the following:

```
*List.renderTable: green, variable
*List.renderTable.green.renditionForeground: Green
*List.renderTable.green.fontName: AS_IS
*List.renderTable.variable.underlineType: SINGLE_LINE
*List.renderTable.variable.renditionForeground: Red
*List.renderTable.variable.fontName: variable
*List.renderTable.variable.fontType: FONT_IS_FONT
*List.renderTable.variable.underlineType: DOUBLE_LINE
*List.renderTable.renditionForegound: black
*List.renderTable.fontName: fixed
*List.renderTable.fontType: FONT_IS_FONT
*List.renderTable.variable.underlineType: NO_LINE
*List.renderTable*tabList: 1in, +1.5in, +3in
```

would set the *renderTable* resource of *List* to a render table consisting of three renditions tagged with **_MOTIF_DEFAULT_LOCALE**, *green*, and *variable* with values for resources set as described in the resource specifications. Note that the *tabList* resource will be shared by all three renditions.

Specifying Font Lists

As was mentioned previously, some of the widgets Motif uses to accept text input from a user do not use renditions. These widgets do not allow a user to change their colors, tabs, or the other miscellaneous properties accessible through renditions. However, these widgets do allow control over the fonts used for text input, although they use a different structure than the render table. These widgets use a structure called a *font list*, containing a list of fonts or font sets, and tags corresponding to each entry.

Font list specification syntax varies depending on whether you want to specify a font or a font set. Font lists are specified in resource files by using the following syntax:

resource_spec : font_entry [, font_entry]+

The resource value string consists of one or more font list entries separated by a, (comma). Each *font_entry* identifies a font or font set and an optional font list element tag. A tag specified for a single font follows the font name and is separated by an = (equal sign). In a font set, the tag is separated by a *:* (colon). The colon is required whether a tag is specified or not. If the font list element tag is not present, Motif assigns the default tag, **FONTLIST_DEFAULT_TAG**.

Use the following syntax to specify a single font for a font entry:

font_name [= tag]

A font entry that specifies a font set is similar, except a; (semicolon) separates multiple font names and the tag is offset with a: (colon). Use the following syntax to specify a font set for a font entry:

font_name [; font_name]+: [tag]

A *font_name* can be an X Logical Font Description (XLFD) string and a *tag* is any set of characters from ISO646IRV, except the space, comma, colon, equal sign, and semicolon.

When specifying fonts, you can provide a detailed description for a given font, or you can use the * (asterisk) wildcard to indicate fields whose values can be selected by the font−loading routines called by an application. The following resource entry specifies a 12−point Courier font that can be of any weight, slant, or set width.

```
*fontList:   -Adobe-Courier-*-*-*--12-120-100-100-*-*
```

The following example specifies a font set for an application running in a Japanese language environment. The font set contains three fonts.

```
*fontList: -JIS-Fixed-Medium-R-Normal--26-180-100-100-C-240;\
           -JIS-Fixed-Medium-R-Normal--26-180-100-100-C-120;\
           -Adobe-Courier-Bold-R-Normal--24-240-100-100-M-100:
```

The default font list element tag is assumed because a tag has not been specified. The font−loading routines will automatically select the character sets required by the locale.

If you encounter unexpected results when specifying a font list resource, see the documentation for the application to determine whether the application places any constraints on how fonts are specified in font lists.

Using Resources for Drag and Drop

Drag and drop resources are generally specified by an application. Refer to the documentation for each application to determine which drag and drop resources you can customize.

Protocol Resources

There are two types of protocols that can be in effect during a drag and drop transaction: preregister and dynamic. Generally, the preregister protocol is more efficient, uses fewer system resources, and provides more attractive drag−over effects. The dynamic protocol provides more attractive and powerful drag−under effects. Motif applications based on Release 2.0 should support both protocols.

You can specify which protocol you want in effect for drag sources and drop sites with the following resources. However, due to other constraints, the system may not be able to use the protocol you requested.

- The *dragInitiatorProtocolStyle* resource requests the protocol style to be used when the client is the initiator of a drag.
- The *dragReceiverProtocolStyle* resource requests the protocol style to be used when the client is a potential receiver of a drop.

These resources can have the following values (the letters in parentheses are used as abbreviations for the values in Table 7−1):

DRAG_NONE (N)

Does not participate in drag and drop.

DRAG_DROP_ONLY (X)

Does not support either the preregister or dynamic protocol, but does allow dropping. This value means that no drag−over or drag−under visual effects will be shown.

DRAG_PREREGISTER (P)

Supports only the preregister protocol.

DRAG_PREFER_PREREGISTER (PP)

Supports both protocols, but prefers the preregister protocol. This value is the default for receivers.

DRAG_PREFER_RECEIVER (PR)

Prefers the protocol that the receiver specifies. This value is only effective for initiators and it is the default for initiators.

DRAG_PREFER_DYNAMIC (PD)

Supports both protocols, but prefers the dynamic protocol.

DRAG_DYNAMIC (D)

Supports only the dynamic protocol.

The following example specifies that an application named *myapp* uses the dynamic protocol both when it is a drag source and when it is a drag receiver.

```
myapp*dragInitiatorProtocolStyle:    DRAG_DYNAMIC
myapp*dragReceiverProtocolStyle:     DRAG_DYNAMIC
```

It may take some experimentation to find the best values for these resources for the applications you are running on your particular system configuration. Here are some guidelines:

- The default protocol style used for the receiver is **DRAG_PREFER_PREREGISTER** and for the initiator is **DRAG_PREFER_RECEIVER**, unless an application has changed it.
- To get the most sophisticated drag−over effects, set the initiator protocol style to **PREFER_PREREGISTER**.
- To get the most sophisticated drag−under effects, set the receiver protocol style to **DYNAMIC** or **PREFER_DYNAMIC**.
- If there are a large number of drop sites, preregistering them all may take unnecessary time. Set the receiver protocol style to **DYNAMIC**.
- If your system is heavily loaded, the **DYNAMIC** protocol may cause unnecessary thrashing. Set the initiator protocol to **PREFER_PREREGISTER**.

- If your system is on a busy network, setting the initiator and receiver protocols to **PREREGISTER** may reduce network traffic.
- If the protocol is set to **DYNAMIC** or **PREREGISTER**, visual effects may be lost if the source and destination have incompatible protocols. Use **PREFER_PREREGISTER** or **PREFER_DYNAMIC** if possible.
- If your system has very limited resources, use the **DROP_ONLY** value to avoid the time and memory requirements of drag−over and drag −under effects completely.

Table 7−1 shows how a conflict between the source and destination protocols is resolved.

Table 1 Source and Destination Protocol Conflict Resolution

Initiator Protocol	*Receiver Protocol*				
	P	PP	PD	D	X
P	P	P	P	X	X
PP	P	P	P	D	X
PR	P	P	D	D	X
PD	P	D	D	D	X
D	X	D	D	D	X
X	X	X	X	X	X
N	N	N	N	N	N

Graphics Resources

The following resources allow you to provide your own default graphics for drag−over situations. These graphics are used only if an application has not specified its own graphics.

defaultSourceCursorIcon

The pixmap used for the source icon.

defaultCopyCursorIcon

The operation icon used when the operation is a copy.

defaultMoveCursorIcon

The operation icon used when the operation is a move.

defaultLinkCursorIcon

The operation icon used when the operation is a link.

defaultInvalidCursorIcon

The state icon used when the drag icon is over an invalid drop site.

defaultValidCursorIcon

The state icon used when the drag icon is over a valid drop site.

defaultNoneCursorIcon

The state icon used when the drag icon is not over a drop site.

You can specify a pixmap and a mask to use for any part of the drag icon by using the following resources:

pixmap

Specifies the pixmap to use.

mask

Specifies the mask for the pixmap, if desired.

Provide the name of the icon you are creating in addition to the resource. For instance, to specify a pixmap and mask for the invalid state icon, use the following resource specifications:

```
*defaultInvalidCursorIcon.pixmap:     mypixmap
*defaultInvalidCursorIcon.mask:       mypixmask
```

You can specify which of the three parts of the drag icon you want to see with the *blendModel* resource. This resource can take the following values:

BLEND_ALL

Use all three parts. This value is the default value.

BLEND_STATE_SOURCE

Use the source icon and the state icon.

BLEND_JUST_SOURCE

Use only the source icon.

BLEND_NONE

Do not display a drag icon.

You can specify how you want the source, state, and operation icon to be located with respect to each other with the *attachment* resource. This resource is used for state and operation icons and ignored for the source icon. The attachment point for these icons is the lower right corner. Use the attachment style that gets the icons closest to the position you want, then use an offset for final adjustment.

The *attachment* resource can take the following values:

ATTACH_NORTH_WEST

Attaches to the upper left corner of the source icon. This value is the default value for the state and operation icons.

ATTACH_NORTH

Attaches to the top of the source icon.

ATTACH_NORTH_EAST

Attaches to the upper right corner of the source icon.

ATTACH_EAST

Attaches to the right side of the source icon.

ATTACH_SOUTH_EAST

Attaches to the lower right corner of the source icon.

ATTACH_SOUTH

Attaches to the bottom side of the source icon.

ATTACH_SOUTH_WEST

Attaches to the bottom left corner of the source icon.

ATTACH_WEST

Attaches to the left side of the source icon.

ATTACH_CENTER

Attaches to the center of the source icon.

ATTACH_HOT

Attaches the icon's hot spot to the source icon's hot spot. This value is best used when the application has provided special source icons just for the purpose of using this value. Otherwise, some odd−looking drag icons may result.

If you specify the *attachment* resource in a resource file, include the name of the icon to which it refers. For instance, to attach the valid state icon at the upper left corner of the source icon, use the following specification:

```
*defaultValidCursorIcon.attachment:  ATTACH_NORTH_NORTHWEST
```

You can move the state and operation icons from the basic attachment point by using offsets. Select the attachment point nearest to the location you want, then make final adjustments with the the following offset resources:

offsetX

The horizontal offset in pixels. Positive is to the right, negative is to the left.

offsetY

The vertical offset in pixels. Positive is down, negative is up.

You can specify how you want the drop site's drag−under visual effects to appear with the *animationStyle* resource. This resource can take the following values:

DRAG_UNDER_HIGHLIGHT

The drop site is outlined when a valid drag is over it. This value is the default.

DRAG_UNDER_SHADOW_OUT

The drop site appears pushed out when a valid drag is over it.

DRAG_UNDER_SHADOW_IN

The drop site appears pushed in when a valid drag is over it.

DRAG_UNDER_NONE

The drop site does not change when a valid drag is over it.

DRAG_UNDER_PIXMAP

The drop site displays a special pixmap when a valid drag is over it.

Use the *animationPixmap*, *animationMask*, and *animationPixmapDepth* resources to specify the pixmap to use if the animation style is **DRAG_UNDER_PIXMAP**.

Color Resources

Use the following resources to set colors for the drag icon:

validCursorForeground

The color to use if the drag icon is over a valid drop site.

invalidCursorForeground

The color to use if the drag icon is over an invalid drop site.

noneCursorForeground

The color to use if the drag icon is not over a drop site.

For example, the following specifications set the drag icon to blue when it is over a valid drop site, to red when it is over an invalid drop site, and to green when it is not over a drop site.

```
*validCursorForeground:     Blue
*invalidCursorForeground:   Red
*noneCursorForeground:      Green
```

Search Paths for Pixmaps and Bitmaps

Many Motif applications display the contents of bitmap and pixmap files. These files are stored somewhere on your file system, typically in directories named *bitmaps*. By default, Motif applications search through a lengthy list of directories in order to find the needed bitmap and pixmap files. For most users, the default search path will work well.

Assuming that the default search path produces good results when you run your Motif applications, you can skip the remainder of this section. If your Motif applications have trouble finding bitmaps or if you just want to learn more about bitmap search paths, read on.

An alternative to using the default search path is to define a search path of your own. The X Window System provides an environment variable named **XBMLANGPATH** for defining your own search path. For example, here is one possible definition for **XBMLANGPATH**:

```
XBMLANGPATH=%B:$HOME/%B:$HOME/bitmaps/%B:/usr/lib/X11/bitmaps/%B
```

As demonstrated in this command, the search path of **XBMLANGPATH** can contain the substitution field *%B*. A Motif application searching for an X bitmap file will substitute the image name for *%B*.

Motif supports the following additional substitution fields:

%N

The class name of the application

%L

The display's language string

%l

The language component of the display's language string

%T

Always mapped to *bitmaps*

%S

Always mapped to NULL

Motif applications use the following rules to search for X bitmap files.

- If **XBMLANGPATH** is set, Motif applications search only through the directories specified by **XBMLANGPATH**.
- If **XBMLANGPATH** is not set but the environment variable **XAPPLRESDIR** is set, Motif applications search for X bitmap files in the following pathnames:
  - %B
  - $XAPPLRESDIR/%L/bitmaps/%N/%B
  - $XAPPLRESDIR/%l/bitmaps/%N/%B
  - $XAPPLRESDIR/bitmaps/%N/%B
  - $XAPPLRESDIR/%L/bitmaps/%B
  - $XAPPLRESDIR/%l/bitmaps/%B
  - $XAPPLRESDIR/bitmaps/%B
  - $HOME/bitmaps/%B
  - $HOME/%B
  - /usr/lib/X11/%L/bitmaps/%N/%B
  - /usr/lib/X11/%l/bitmaps/%N/%B
  - /usr/lib/X11/bitmaps/%N/%B
  - /usr/lib/X11/%L/bitmaps/%B
  - /usr/lib/X11/%l/bitmaps/%B
  - /usr/lib/X11/bitmaps/%B
  - /usr/include/X11/bitmaps/%B

- If neither **XBMLANGPATH** nor **XAPPLRESDIR** is set, Motif applications search through the following default search paths:
  - `%B`
  - `$HOME/%L/bitmaps/%N/%B`
  - `$HOME/%l/bitmaps/%N/%B`
  - `$HOME/bitmaps/%N/%B`
  - `$HOME/%L/bitmaps/%B`
  - `$HOME/%l/bitmaps/%B`
  - `$HOME/bitmaps/%B`
  - `$HOME/%B`
  - `/usr/lib/X11/%L/bitmaps/%N/%B`
  - `/usr/lib/X11/%l/bitmaps/%N/%B`
  - `/usr/lib/X11/bitmaps/%N/%B`
  - `/usr/lib/X11/%L/bitmaps/%B`
  - `/usr/lib/X11/%l/bitmaps/%B`
  - `/usr/lib/X11/bitmaps/%B`
  - `/usr/include/X11/bitmaps/%B`

  These paths are defaults that vendors may change. For example, a vendor may use different directories for `/usr/lib/X11` and `/usr/include/X11`.

The contents of the file must conform to the rules for X11 bitmap and pixmap files. In other words, Motif can read any X11 conformant bitmap or pixmap file.