Using Customization Features

Motif allows you to specify your preferences, for visual characteristics, such as colors and fonts. You can also specify certain behaviors, such as the response to a keystroke or mouse button action. These characteristics are established in configuration files that are read by Motif applications each time they start up. To specify your preferences you can modify the configuration files or you can override the specifications in the configuration files by setting options in the command line that starts an application.

This chapter provides the following general information on customization:

- Understanding the resource database
- Resource specification syntax
- Modifying the resource files
- Setting resources on the command line
- Using the **xrdb** client program
- Using internationalized applications

Chapters 5, 6, and 7 describe in detail how to customize **mwm** and Motif applications.

Application Resources

Many aspects of an application's appearance and behavior, such as location on the screen, color, and size, are controlled by sets of variables called *resources*. Resources are assigned values that determine specific characteristics of an application. Depending on a resource's function, it can take values that are either names, numbers, or Boolean values. For example, if the resource sets the color of an application or part of an application, you specify a predefined color string value as the resource's value, such as *DarkSlateBlue* or *Black*. If the resource sets the title line of an application, you can specify a name string of choice. You can also set the resource to a numeric value for operations like defining the location of the application. Some resource values are set with values of either **True** or **False**.

Application resources take values that can be set a number of different ways. Resources whose values are not set are assigned default values. Each application builds an initial database of resources based on the supplied and default values.

How Application Resource Databases Are Set Up

The setting of an application's resource database is a linear process. Different values for the same application resource(s) can be set at different times before an application is actually started up. The resource database is updated as resource values are changed.

The value that is finally set for a resource depends on when during the hierarchical process that particular value was being set. However, the setting of an application's resources are not dynamic. Resource values that are set after an application has already been started do not take effect until that application is restarted.

The application resource database is set up according to the following hierarchy, from the highest to lowest precedence:

- The application command line
- A per−host user environment resource file on the local host
- The screen−specific resources for the screen
- The resource property on the server or a user preference resource file on the local host
- An application−specific user resource file on the local host
- An application−specific class resource file on the local host

For example, if the color of an application was set to *Blue* in an application−specific user resource file on the local host, but you specified, during the command−line invocation, that this application's color be set to *Red*, then the same resource was defined for two values, with the command −line resource setting of *Red* overriding the earlier setting of *Blue*. Thus, *Red* is registered in that application's resource database. Note that most applications have default values to which resources are set if you do not specify a value for a particular resource.

As you can see from the listed hierarchy, applications often set up their database of resource values using existing resource files. Some of these resource files come with the system, and some are files that you create yourself.

The following subsections describe in more detail the different ways that an application's resource database can be set.

Setting Application Resources from the Command Line

Most applications provide command−line options that allow users to set resource values. These options, listed in Table 4−1, provide customization for the most common resources.

Table 1 Some Standard X Command−Line Options

Option	*Abbreviation*	*Description*
–background	–bg	Color of the window's background
–bordercolor	–bd	Color of the window's border
–borderwidth	–bw	Width of the border in pixels
–display	–d	Display on which the client can run
–foreground	–fg	Color of the text or drawings
–geometry	–g	Size and placement
–iconic	–i	Start application as an icon
–name	–n	Name of the application
–reverse	–rv, +rv	Reverse foreground and background colors
–title	–t	Text displayed in the title bar
–xrm	None	Next argument contains any resource specification

In most cases, these options will correspond to specific resources. For example, if you use the –background option to change an application's background color, the *background* resource value is set to that color. Note, however, that option names are not always the same as the resource they are setting values for.

Setting resource values at the command line is useful to set up an application in a particular way just this once. However, if you intend to use the same resource setting for that particular application, you should put the resource specification into your user–preferred resource file (typically `.Xdefaults` in your home directory).

Per–Host User Environment Resource Files

Next in the hierarchical priority is the setting of user– and machine–specific resources. At initialization, the application checks if the **XENVIRONMENT** environment variable has been set to the name of a specific resource file. If one has been set, then the application gets resource value settings from this file. If the environment variable has not been set, then the application initialization operation looks in the file `$HOME/.Xdefaults-`*host*, where *host* is the name of the host or machine running the application. This is a resource file that you can set up to run on your particular machine.

Per–Screen Resources

As of Release 5 of the X Window System (X11R5), any per–screen properties can be set and stored in the **SCREEN_RESOURCES** property of the root window of the default screen of the application's display. For example, you can specify color resources for a color screen and monochrome resources for a monochrome screen. A resource database is created for each screen, and the application finds the resources that are appropriate for each screen. The database is used to define resources when multiple screens are in use and, if some resources are not defined for all screens, it is typically set up with the `xrdb -screen` or `xrdb -screens` command. The former sets the **SCREEN_RESOURCES** property for the display's default screen, the latter for a display's multiple screens. Note, however, that when the system needs resources for a screen that is not the display default, it uses the **SCREEN_RESOURCES** property of that other screen instead of the display's default.

Server or User–Preference Resources

General resources that apply to all the screens of a display can have these common resources set and stored in the **RESOURCE_MANAGER** property of the root window on the default screen of the display. You can also use this property to set the resources of a screen if it is the only screen of a display. This property is typically set up with the `xrdb -global` command if some resources are not defined for all screens. If this property does not exist, the contents of the file `.Xdefaults` in your home directory are used instead. Your personal `.Xdefaults` file is the resource file where you usually put resource specifications that customize applications for your own use.

User Application File

You can also have your own application–specific resource file to set resources. This file sets resources for just one particular application, typically for one user or group of users. Usually, the name of the application–specific resource file is listed in that application's associated

reference page, and is the name of that application's class. As an example of an application−specific resource file name, the application−specific resource file for the **xcal** application is named **XCal**.

When an application looks for an application−specific resource file, it first checks the **XUSERFILESEARCHPATH** environment variable, which controls the search path that applications use to find application−specific resource files, then the **XAPPLRESDIR** variable, then an implementation−dependent predefined search path starting from the user's home directory. Refer to Section 4.1.1.7 for a description of file search paths.

Application Class Resource File

To set all the resources of a particular application class, an application−specific class resource file should be used. Application class resource files are application−specific default files that are defined for the entire system. However, instead of being named with the application name, they are named the according to the application's class name. For example, the **xcal** application has a class name of **XCal**, which would also be the name of its application−specific default file. Application default files are typically predefined and not generally accessible to the user. But, you can check these files to see what the default values are for particular applications, then change them in your own user−specific resource files or on the command line.

When an application looks for an application−specific class resource file, it first checks the **XFILESEARCHPATH** environment variable for guidance, then it uses an implementation−dependent predefined search path.

File Search Paths

An application searches for resource files and any localized databases on a file search path. A file search path is an ordered set of pathnames. If a resource file is not found in the first location, the application searches in the next location, and so on, until a resource file is found or all of the locations have been searched.

File search paths can incorporate a set of substitution characters that represent variable data. At run time, an application looks at certain external data and supplies corresponding values for each substitution character in the file search path. X applications accept the following substitution characters:

- %*N* is replaced by the class name of the application.
- %*C* is replaced by the value of the *customization* resource when searching for application defaults files.
- %*L* is replaced by the display's language specification. The format of the language specification is implementation dependent; it may have language, territory, and codeset components.
- %*l* is replaced by the language part of the language specification.
- %*t* is replaced by the territory part of the language specification.
- %*c* is replaced by the codeset part of the language specification.
- %% is replaced by %.

For example, *%N* can represent the class name of an application. When *%N* is in a path description, the application class name is substituted into the pathname. If the first path in a search path hierarchy is `/usr/lib/X11/app-defaults/%N` and the application class name is *Mailer*, the *%N* substitution causes the application to look in the directory `/usr/lib/X11/app-defaults` for a file named *Mailer*, which contains its application class defaults.

There are a number of environment variables that you can set to specify the search paths that an application uses to find various resource files. For example, the **XUSERFILESEARCHPATH** environment variable controls the search path that applications use to find your application −specific resource files. You can also set the **XENVIRONMENT**, **XFILESEARCHPATH**, and **XAPPLRESDIR** environment variables to specify search paths for other aspects of resource lookup.

For more information about these features, refer to the *Programmer's Supplement for Release 5*, 1991, published by O'Reilly and Associates, Inc.

Resource Specification Syntax

To understand how to assign a resource value, a description of the resource specification syntax is first provided. You assign resource values through this syntax, especially when using resource specification files.

An application's resource specification syntax is composed of the name of the application (or client), its child widgets, and the resource that is being specified. It depends on how the components of an application are organized; in other words, the hierarchical relationship between the parent and child widgets. For example, in a mail program there might be a MainWindow containing several PushButtons. Each PushButton might have an associated subwindow that contains any number of Menus. If you want to specify the font size for a single Menu in one of the subwindows, you need to know the names of all the widgets and their positions in the widget hierarchy.

These hierarchical components of a resource specification syntax are separated by one of three separators.

.

The dot is used between either an instance or class name and the lower widgets of a hierarchy to indicate a tight binding, where the presented hierarchy must be in the correct order. Usually the name of a class or an instance of the application comes first, followed by the name of the highest widget in the widget hierarchy. The name of the top widget in the hierarchy is followed by the specification of any number of widgets lower in the hierarchy. Finally, the resource to be specified is placed at the end of this hierarchical specification.

*

The asterisk is used between either an instance or class name and any of the lower widgets of a hierarchy, or the resource itself, to indicate a loose binding, where the asterisk acts as a wildcard. The asterisk substitutes for any number of components in the widget hierarchy. It can also be used as the first component in a resource specification, without an preceding instance or class name.

Note that using the * (asterisk) causes the system to assume that the resource value being set is to apply to all the represented components. Because of this capability, you should be careful of unexpected results from overuse of the * (asterisk). For example, if you specify a resource specification of the following:

```
*Foreground:    Purple
```

this would make the foreground of any client/application that had a foreground resource be purple.

?

Unlike the preceding two separators, the question mark is used between two dots, and substitutes for a single widget or class name. It can also substitute for the first component in the resource specification, be followed by an asterisk, and not have a preceding period.

A resource specification syntax can be expressed by using any combination of these separators.

Every application and resource in X has both an instance name and a class name. The instance name identifies each application and resource individually. The class name specifies the general category to which each individual instance of an application or resource belongs. For example, the class name *Mailer* specifies all instances of an application called *mailer*. Class names always begin with an uppercase letter and instance names always begin with a lowercase letter. If the instance name of a specific component is a compound word, like *pushButton*, the second word usually begins with an uppercase letter. The class name and the instance name for an application are often the same, except for the case of the initial character(s). The instance name for an application is usually the name of the command that is used to start the application. The specification of an instance and its classes can be a hierarchical one, where an instance can have multiple classes.

The presence of the ? (question mark) separator slightly changes the existing X11 rules for matching components when an application looks up the value for a resource. When a resource match is requested, the system performs a left–to–right scan of the resource specification supplied by the application. When there is some ambiguity in the matching process, such as that caused by use of one of the wildcards, the following rules are followed to determine the priority order for specifying the resource value:

- resource match by the current component's name has priority over a match with the same component by class. A match by either component name or class takes precedence over a match by the ? (question mark) substitution character, and any of the three matches has priority over an * (asterisk) substitution.
- resource that uses a. (dot) to precede the current component has a higher priority than one that uses an * (asterisk) in the same position.

Using #include in Resource Files

When using the **xrdb** client program, resource files can also use the `#include "`*filename*`"` directive to include other resource files into the

current resource file. This can be used to reduce duplicate entries in resource files. For example, if there is a base resource file for an application, and you simply want to reset the values of some of the base resources for this application, you can simply use `#include` to put the file into your own application–specific resource file and change just the resource values you want to change, leaving the other resources as they are.

Assigning Resource Values

This section provides examples of how to use the resource specification syntax. In general, these syntaxes are most important when used in resource specification files. Most of the examples illustrate typical lines you might use in a resource specification file. However, as shown in Section 4.3, if you are using an application's command line to set resource values, using fuller resource specifications in the command line can ensure that these settings override existing ones.

The following is a format that uses a. (dot) as a separator.

Client . widget1 . widget2 .... resource : value

As an example, you could use the following specifications in your `.Xdefaults` file to set resources for **xcal** clients on your system:

```
XCal.edit.geometry: 350x200
```

In this example, the specification states that only the geometry resource for the edit child widget of the **XCal** parent widget is to be set with the size of 350 by 200 pixels. **XCal** is the class name.

The following is a format that uses an * (asterisk) as a separator. It illustrates a general, abbreviated format:

Client * resource : value

As an example, you could use the following specifications in your `.Xdefaults` file to set resources for **xterm** clients on your system:

```
XTerm*background:       Wheat
XTerm*foreground:       Navy
XTerm*font:             fixed
XTerm*scrollBar:        true
Xterm*geometry:         80x30
```

In this example, every instance of an **xterm** window on your system appears with a wheat–colored background and navy–colored foreground (the color of any text or graphics that appear in the window), uses the font named *fixed*, has a size of 80 by 30 pixels, and provides a ScrollBar. **XTerm** is the class name.

When using an * (asterisk) to substitute for the class name, the resource specification line would appear as the following:

```
*Foreground: Blue
```

This specification ensures that the foreground (text and graphics) in all clients will be blue.

You can use the following specification in your `.Xdefaults` file to set the font for every window in every instance of the *mailer* application:

```
Mailer*fontList: fixed
```

The next example shows how you would specify the font only for the area in which you compose mail messages in the *mailer* application:

```
Mailer*messageArea*fontList: fixed
```

The following is a format that uses the ? (question mark) separator.

Client . ? . resource : value

As an example, you can use the following resource specification to set the background color for all of the widgets that are grandchildren of the top widget in the hierarchy for the application:

```
Mailer.?.?.Background: Red
```

With X11R5 and the **xrdb** client program, you can now also use C preprocessor commands in your `.Xdefaults` file to specify per–screen resources. For example, you can separate resource specifications for color screens from monochrome screens as follows:

```
#ifdef COLOR
*Background: Grey
*Foreground: Navy
#else
*reverseVideo: True
#endif
```

You can also specify unit types in your `.Xdefaults` file for any resource that is of type Dimension or Position with the following format:

<floating value><unit>

where:

unit

is <"", *pixels*, *inches*, *centimeters*, *millimeters*, *points*, *font units*>

pixels

is <pix, pixel, pixels>

inches

is <*in*, *inch*, *inches*>

centimeter

is <*cm*, *centimeter*, *centimeters*>

millimeters

is <*mm*, *millimeter*, *millimeters*>

points

is <*pt*, *point*, *points*>

font units

is <*fu*, *font_unit*, *font_units*>

float

is {"+"|"−"}{{<"0"−"9">*}.}<"0"−"9">*

Note that the type Dimension must always be positive.

For example,

```
xmfonts*XmMainWindow.height: 10.4cm
*PostIn.width: 3inches
```

The documentation for an application should provide the instance and class names of any components that the application allows you to customize. When appropriate, use the class name in a resource specification to ensure that all instances will use the same resource values. Chapters 5, 6, and 7 provide additional information about the specification of particular resources for **mwm** and Motif applications.

Using Command–Line Options

Many X applications provide the same basic set of options, as listed in Table 4–1. Often, you can just supply simple arguments to these options to have them set the resources properly. Other times, you need to define fuller resource specifications to get the values to override the existing resource settings. This section provides some examples of how to use command–line options to set resource values.

The simplest use of a command–line option is to pass it a single argument, as follows. If you want to start a new terminal window with a color that is not the background color specified in the configuration file, start the client by using the –bg option; all other resources will take the default values. For example:

```
% xterm –bg Red &
```

In the case of valid resources that do not have associated command–line options, you can use the –xrm option to set the values. Any valid resource specification can follow the –xrm option in '' (single quotes). For example:

```
% xclock –xrm '*hands: Red' &
```

Resources specified in this manner are only applicable to the current instance of the program.

In the preceding examples, a more specific resource specification was unnecessary to achieve the desired effect. However, a resource specification in a configuration file can take precedence over a command–line option if the command line is less explicit. For example, if the resource file includes this specification:

```
xclock*hands: Blue
```

the following command line will have no effect:

```
% xclock –xrm '*hands: Red' &
```

To override the specification in the resource file, you would have to enter the following command line:

```
% xclock –xrm 'xclock*hands: Red' &
```

Note that not all Motif applications support all of these command–line options. To see which options are actually available for a particular application, refer to the documentation for the application.

Modifying the Resource Files

Most of the features that you will want to customize can be set with resources in the `.Xdefaults` file in your home directory. However, key bindings, mouse button bindings, and menu definitions for **mwm** are specified in the supplementary `.mwmrc` file, which is referenced by resources in the `.Xdefaults` file.

If you do not have a `.mwmwc` file in your home directory, you can copy it as follows:

```
% cp /usr/lib/X11/system.mwmrc   ~/.mwmrc
```

If you do not have a `.Xdefaults` file in your home directory, you can create one with any text editor. Once you have these files in your home directory, you can set resource values in them as you wish. Because you've made your own copy of the the `.mwmrc` file, your specifications will not interfere with the specifications of other users.

Using the xrdb Client Program

The X Window System uses the **xrdb** client program to create and update a resource database on each server. Normally, **xrdb** is run when X starts up. If you make any changes to resource settings in the resource files, you can run **xrdb** for the new changes to take effect. Note that the **xrdb** is an optional utility for the setting up of the resource database.

To replace old resource settings with new settings, run **xrdb** with the −load option, as shown in the following example:

```
% xrdb .Xdefaults
```

In this example, any changes you made to resource specifications in the `.Xdefaults` file are available to clients that use these resources. Any resource that you do not specify a new value for will be overwritten with an empty value. After replacing resource settings in the database, you need to restart any clients, including **mwm**, so that they will reread the database and use the new resource settings.

To append new settings without changing the old settings, run **xrdb** with the −merge option, as shown in the following example:

```
% xrdb −merge values.new
```

In this example, the resource settings contained in the file `values.new` will be appended to the resource database. You can specify the changes in a file, as in this example, or you can specify an individual resource setting, as shown in the following example:

```
% xrdb −merge
xterm*scrollBar: True
```

Press `Ctrl-D` to signal the end of input.

To check the current resource settings, run **xrdb** with the −query option. If you have not previously run **xrdb**, you will not get any output from this command. If you have run **xrdb**, the output will look something like the following example:

```
xterm*scrollBar: on
xterm*font: terminal14
Mwm*keyboardFocusPolicy: explicit
Mwm*buttonBindings: ExplicitButtonBindings
emacs*geometry: 77x34x17+225
```

As of Release 5 of the X Window System, **xrdb** has been modified so that you can now specify resources on a per−screen basis. The −global, −screen, −screens, and −all options control whether resources are read from the screen−independent property, from screen−dependent properties, or both. The −remove option indicates that the specified property should be removed from the server. For more information about these options, as well as many other options for advanced customization, see the reference pages for **xrdb** in the X Window System documentation.

Using Internationalized Applications

Internationalization is a method of application development that enables an application to be used in a variety of *locales*. The concept of a locale is used to encompass the characteristics and requirements of a given language. An internationalized application requires that any language −dependent or custom−dependent information be stored external to the application program. A locale can be characterized by several components. For example, character sets, sort order, text direction, and formats for data such as date and currency are used to describe the unique characteristics of a locale. For each different locale, information such as menu items, help information, and user prompts is defined and stored separately. The information is localized, which means that it has been tailored for a specific language and/or country.

An application designed to run in different locales examines certain resources and environment variables in order to determine which language to use when executed. An application establishes file search paths for resource files and other language−dependent information. There is a wide variety of information that can be localized. Localized information is stored in files that reside in different directories. Motif uses an underlying *Xt* Intrinsics mechanism (**XtResolvePathname**) to select and locate the appropriate files, depending on the locale.

An internationalized application can set its locale at run time, typically using an internal procedure called a *language procedure*. The language procedure processes data in your user environment and sets the application's locale. This feature enables you to modify your user environment and rerun an application in a different locale. An application can use a default language procedure or supply its own procedure. As a result, two applications can process the same user environment data with different results.

Refer to the user documentation for the application you intend to use for detailed instructions on using its internationalized features. For more detailed information about internationalization issues, refer to the "Internationalization" chapter in the *Motif Programmer's Guide* and the *Programmer's Supplement for Release 5*, 1991, published by O'Reilly and Associates, Inc.

Layout Direction

Layout direction is the direction that is used to display visual components such as PushButtons, PopupMenus, ScrollBars, titles, and so on. In general, this direction matches the direction that people use when reading or writing in a particular language. Languages such as English, French, German, and Swedish are read and written from left to right, top to bottom. Therefore, when users working in those languages enter characters from a computer keyboard, each new character is displayed to the right of the preceding one. These same users would also expect the layout of other visual components to be displayed from left to right. For example, when you use a Pulldown Menu to access a cascading menu in a left−to −right environment, the cascading menu pops up to the right of the Pulldown Menu. Another example is the way that ToggleButtons are laid out. In a left−to−right environment, the ToggleButton is located to the left of its text label. However, for right−to−left languages, such as Arabic or Hebrew, the ToggleButton could be located to the right of its text label. The layout direction of a visual component is defined by the application, and is affected by the environment's locale. In the example of the cascading menu, the menu would cascade to the left.

In some Asian languages, text is drawn vertically. Vertical writing is enabled when the **XmText**resource **XmNlayoutDirection** is set to **XmTOP_TO_BOTTOM**. In addition to causing text that you enter to be drawn vertically, the **XmText** widget exhibits other behaviors consistent with this manner of entering and editing text; for example,

- If word wrapping is turned on, text is wrapped from one column to the next, rather than from line to line.
- A key binding that (in the case of English) would cause the cursor to move to the next character in the current line, instead causes the cursor to move to the next character in the current column.

Input Methods

An *input method* is the underlying mechanism that takes keyboard input and displays the locale's corresponding character(s) on the screen. An input method interprets the user's keystrokes based on the conventions supplied by the input method. The input method used by an application based on an alphabetic language is generally invisible. However, ideographic languages, such as Chinese or Japanese, may use an input method that composes keystrokes in a separate window called a *pre−edit window*. For example, you type a phonetic representation of a spoken word and the input method determines the ideographic character that is pronounced in that way. When more than one character meets the criteria, the input method displays a list of characters to select from. Once you confirm a selection in the pre−edit area, the information is passed to the application.

Input methods can be defined by the platform vendor, an application, or a user. Information about available input methods and their features can be found in the documentation for the system or application that you are using. Motif provides support for the following input method styles: **OffTheSpot**, **Root**, **None**, **OverTheSpot**, and **OnTheSpot**.

For example, when the **OnTheSpot** input style is used, a pre−edit string is displayed in the text widget window. Depending on the text insertion mode, new text is inserted into the existing text or overwrites it. You can edit the pre−edit string until you commit the buffer by some action such

as cutting, pasting, selecting an object, moving the cursor, or using a "commit" key. For a given application, the action that cause the string to be committed depends on the implementation of the input method for that application.

Setting and Modifying the Language Environment

An internationalized application can dynamically set its language environment when it is run. The design of the application defines what information is expected and how it is used to set the language environment. Typically, you can modify the language environment used by an application by specifying a language resource or a language environment variable. The exact method that you use to set the language environment for an application is defined by the application, so you need to see the documentation for the application for specific information.

Message Catalogs

Some internationalized applications use message catalogs to display text to users. Message catalogs are files that store text that is particular to specific locales (for example, fr_FR.ISO8859−1). Refer to Chapter 5 for more information on these catalogs.