11.4 Resource Specification and Matching

A resource specification consists of an optional name of a client, followed by one or more predefined variables that indicate the preference to set, followed by a colon, optional white space, and the actual value of the preference.*

The format of these preference strings is most easily seen by looking at a resource database file, such as the one shown in Example 11-3.

Example 11-3. A simple resource database file

```
*font:                    fixed
.borderWidth:             2

xterm.scrollBar:          on
xterm.title:              xterm
xterm.windowName:         xterm
xterm.boldFont:           8x13
xterm.curses:             off
xterm.internalBorder:     2
xterm.iconStartup:        off
xterm.jumpScroll:         on
xterm.reverseWrap:        true
xterm.saveLines:          700
xterm.visualBell:         off
```

*Thanks to Jim Fulton of the X Consortium for providing an explanation of the resource manager on the *comp.window.x* network news group, from which portions of this section are excerpted.

The options which begin with a period apply to all programs unless overruled by a program-specific entry with the same resource name. The last element between a period or asterisk and the colon is the resource name.

This simple example demonstrates the rules as commonly practiced in Xlib applications. However, there are a number of additional rules that come into play in more complex, object-oriented applications, such as those written with the Xt Toolkit. Preferences may apply only to a particular subwindow within an application. For example, the *xmh* mail handler allows the user to set preferences for multiple levels of windows. These levels can be specified explicitly or by using a wildcard syntax, denoted by the asterisk.

As a result, you should think of the syntax for preference specifications, not as:

client.keyword: value

but as:

object . . . subobject . . . resourcename: value

where the hierarchy of objects and subobjects not only usually corresponds to major structures within an application (such as windows, panels, menus, scrollbars, and so on) but also can be a class of such objects.

Individual elements in the hierarchy of objects and subobjects are called *components*. Component names can be either *instance names* or *class names*. By convention, instance names always begin with a lower-case letter, while class names always begin with an upper-case letter. Instances and classes are concepts in object-oriented programming, not normally used in Xlib programming. For a detailed description of instance and classes and how they appear in resource specifications, see Volume Four, *X Toolkit Intrinsics Programming Manual*.

Both instance and class names may include either upper-case or lower-case letters anywhere but in the starting position; in fact, for clarity, a component name is often made up of multiple words concatenated without spaces, with an initial capital serving as the word delimiter. For example, `buttonBox` might be the instance name for a window containing command buttons, while `ButtonBox` would be the corresponding class name.

For example, consider a hypothetical mail-reading program called *xmail*, which is similar to the current *xmh* application.* As shown in Figure 11-3, *xmail* is designed in such a manner that it uses a complex window hierarchy, all the way down to individual command buttons which are small subwindows. If each window is properly assigned a name and class, it becomes easy for the user to specify attributes of any portion of the application.

The top-level window is called `xmail`. It contains a series of vertically-stacked windows (panes), one of which contains all the command buttons controlling the program's functions. This control pane is named `toc` (table of contents). One of the command buttons is used to incorporate (fetch) new mail.

*We do not discuss the actual *xmh* application, even though it does use an object-oriented approach, because speaking hypothetically gives us greater freedom to set up illustrative examples.

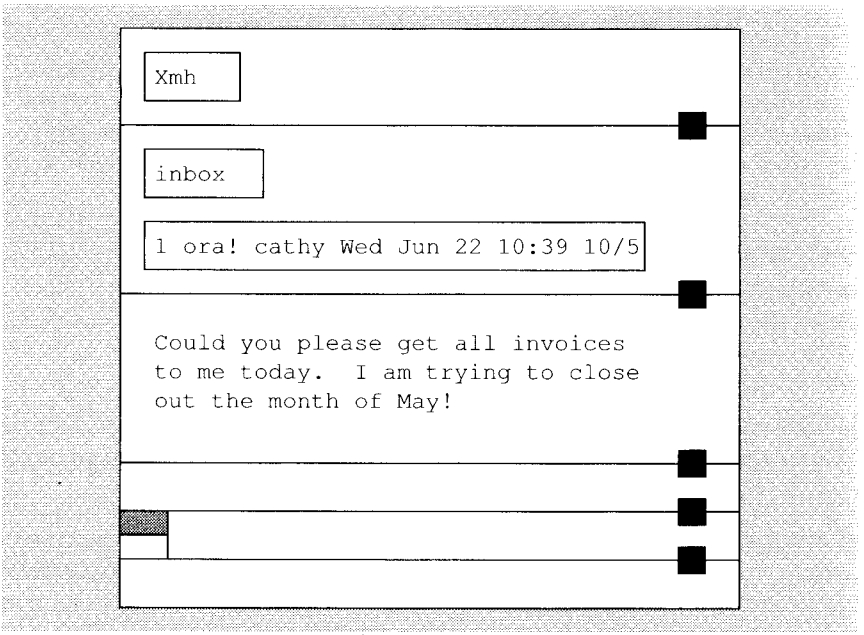

Figure 11-3. The hypothetical xmail display

This button needs the following resources:

- Label string
- Font
- Foreground color
- Background color
- Foreground color for its highlighted state
- Background color for its highlighted state

A full instance name specifying the background color of the include button might be:

```
xmail.toc.includeButton.backgroundColor
```

Defining class names allows the user to set resource values more freely. The pane containing the buttons could be of class `ButtonBox`, and the buttons themselves are all of class `CommandButton`. Therefore, the background of all the buttons could be identified with:

```
Xmail.ButtonBox.CommandButton.BackgroundColor
```

The user could do something like this:

```
Xmail.ButtonBox.CommandButton.BackgroundColor:  blue
xmail.toc.includeButton.backgroundColor:        red
```

which would make all command buttons blue except the one instance specified (`includeButton`), which would be red.

It might not be immediately apparent how you could use a class for the name of the application itself. However, consider the emacs family of text editors. Microemacs and GNU emacs could both be considered members of the class `Emacs`. Or assume that you were using a toolkit to build a group of applications with a similar user interface. It might be desirable to let the user specify attributes for features of all these applications. You might define a general class of vertically paned applications called `Vpane`.

The distinction between instance names and class names becomes important when you are retrieving an option value from the resource database. Routines such as `XrmGetResource` that retrieve data from the database must specify two separate strings (*retrieval keys*)—one made up completely of instance names, and the other of class names. Both strings must be fully specified.

Tight Bindings and Loose Bindings

The components in a resource specification can be bound together in two ways: by a tight binding (a dot, .) or by a loose binding (an asterisk, *). Thus, `xmail.toc.background` has three name components tightly bound together, while `Vpane*Command.foreground` uses both a loose and a tight binding.

Bindings can precede the first component but may not follow the last component. By convention, if no binding is specified before the first component, a tight binding is assumed. For example, `xmail.background` and `.xmail.background` both begin with tight bindings before the `xmail`, while `*xmail.background` begins with a loose binding.

The difference between tight and loose bindings comes when a function like `XrmGetResource` is comparing two resource specifications. A tight binding means that the components on either side of the binding must be sequential. A loose binding is a sort of wildcard, meaning that there may be unspecified components between the two components that are loosely bound together. For example, `xmail.toc.background` would match `xmail*background` and `*background`, but not `xmail.background` or `background`.

Because loose bindings are flexible, they are very useful for defining resource specifications. They allow resource specifications to match many specific applications and will still match if the applications are slightly changed (for example, if an extra level is inserted into the hierarchy.)

A resource specification used to store data into the database can use both loose and tight bindings. This allows the user to specify a data value which can match many different retrieval keys. In contrast, retrieval keys from the database can use only tight bindings. You can only look up one item in the database at a time.

Remember also that a resource specification can mix name and class components, while the retrieval keys are a pair of specifications without values, one consisting purely of name (first character lower-case) components and one consisting purely of class (first character upper-case) components.

11.4.2 The -name Option

If you set up your resource specifications to use the class name for a program instead of an instance name, users can then list instance resources under an arbitrary name that they specify with the *-name* option to a program. For example, if *xterm* were set up this way, with the following resources defined:

```
XTerm*Font:                6x10
smallxterm*Font:           3x5
smallxterm*Geometry:       80x10
bigxterm*Font:             9x15
bigxterm*Geometry:         80x55
```

the user could use the following commands to create *xterms* of different sizes:

```
xterm &                    (create a normal xterm)
xterm -name smallxterm     (create a small xterm)
xterm -name bigxterm       (create a big xterm)
```

11.4.3 Storage/Access Rules

The algorithm for determining which resource name or names match a given query is the heart of the database. The idea is that resources may be stored with only a partially specified instance and/or class name, while they are accessed with a completely specified instance and class name pair. The lookup algorithm then searches the database for the instance that most closely matches this full instance and class name pair.

The resource manager must solve the problem of how to compare the pair of retrieval keys to a single resource specification. (Actually, it must compare the pair of retrieval keys to many single resource specifications, since the resource manager will compare the retrieval keys against every resource specification in the database, but one at a time.)

The solution of comparing a pair of keys to a single resource specification is simple. The resource manager compares component by component, matching a component from the resource specification against both the corresponding component from the instance retrieval key and the corresponding component from the class retrieval key. If the resource specification component matches either retrieval key component, then that component is considered to match. For example, the resource specification `xmail.toc.Foreground` matches the instance retrieval key `xmail.toc.foreground` and the class retrieval key `Vpane.Box.Foreground`.

Because the resource manager allows loose bindings (wildcards) and mixing names and classes in the resource specification, it is possible for many resource specifications to match a single instance/class retrieval key pair. To solve this problem, the resource manager uses the following precedence rules to determine which is the best match (and only the value from that match will be returned). The precedence rules are, in order of preference:

1. The attribute of the name and class must match. For example, assuming that Xterm allowed multiple levels of resource specification, queries for:

```
xterm.scrollbar.background      (name)
XTerm.Scrollbar.Background      (class)
```

will not match the following database entry:

```
xterm.scrollbar:  on
```

2. Database entries with instance or class prefixed by a dot (.) are more specific than those prefixed by an asterisk (*). For example, the entry `xterm.geometry` is more specific than the entry `xterm*geometry`.

3. Instances are more specific than classes. For example, the entry `*scrollbar.background` is more specific than the entry `*Scrollbar.Background`.

4. An instance or class name that is explicitly stated is more specific than one that is omitted. For example, the entry `Scrollbar*Background` is more specific than the entry `*Background`.

5. Left components are more specific than right components. For example, the entry `xterm*background` is more specific than the entry `scrollbar*background`.

As an example of these rules, assume the following user preference specifications:

```
xmail*background:                          red
*command.font:                             8x13
*command.background:                       blue
*Command.Foreground:                       green
xmail.toc*Command.activeForeground:        black
xmail.toc.border:                          3
```

A query for the name:

```
xmail.toc.messageFunctions.include.activeForeground
```

and class:

```
Vpane.Box.SubBox.Command.Foreground
```

would match `xmail.toc*Command.activeForeground` and return "black." However, it also matches `*Command.Foreground` but with lower preference, so it would not return "green."

The programmer should think carefully when deciding what classes to use. For example, many text applications have some notion of background, foreground, border, pointer, and cursor or marker color. Usually the background is set to one color, and all of the other attributes are set to another, so that they may be seen on a monochrome display. To allow users of color displays to set any or all of them, the colors might be organized into classes as follows:

Table 11-1. Setting Classes

Instance	Class
background	Background
foreground	Foreground
borderColor	Foreground
pointerColor	Foreground
cursorColor	Foreground

Then to configure the application to run in "monochrome" mode but using two colors, the user would only have to use two specifications:

```
obj*Background:  blue
obj*Foreground:  red
```

Then if the user decided to make the cursor yellow but have the pointer and the border remain the same as the foreground, you would only need one new resource specification:

```
obj*cursorColor: yellow
```

All the resource manager rules for matching and precedence are explained in more detail in Volume Four, *X Toolkit Intrinsics Programming Manual*.