OLIT Resources 3

OLIT resources are named data units that specify widget attribute values. Widget attributes are specified in resource/value pairs and set characteristics such as widget color, font, layout, alignment, and label. Resources are set in five ways:

- In the application code when the widget is created
- In the application code after the widget is created
- Through the resource database
- In a command line option
- Dynamically while the application is running

If a resource is not set in any of these ways, OLIT will set the resource to a default value. This chapter describes setting resources in the source code and in a resource file. Setting resources in the command line or dynamically is described in *The X Window System Programming and Applications with Xt*.

OLIT Resource Documentation

Widget class resources are documented alphabetically in the *OLIT Reference Manual*. Widget class-specific resources are listed and described in the section about the widget class. C*ommon resources*, resources that widget classes inherit from their widget class parents (see Figure 1-3), are described in Chapter 2 of the OLIT Reference Manual.

Common Resource

Because all widgets are subclassed from the *Core* widget class, all widgets inherit the Core resources. Likewise, widget classes such ControlArea and Rubbertile are further subclassed from *Composite* and *Manager* and so inherit the resources of these two classes. Common resources are described in Chapter 2 of the *OLIT Reference Manual*.

While widget classes may share many common resources, how a common resource affects one widget class may differ in the way it affects another widget class. These differences, along with the widget class's own unique resources, are documented in the specific widget class sections in the *OLIT Reference Manual*.

The *OLIT Reference Manual* lists resources by widget class, including the resources of all its superclasses, to reflect the OLIT architecture. Thus, the ControlArea Resource Table lists the ControlArea resources under four widget classes, *Composite*, *Core*, *Manager,* and *Widget*. In most cases, however, you aren't interested in the architecture, but simply wish to know what the settable resources are, what attributes do they control, what type of values do they take, and what the default is. We provide this information along with the architecture information so you will understand the relationship and shared characteristics of OLIT widgets.

Setting Resource Values at Widget Creation

`XtVaCreateManagedWidget()` allows you to set widget resource values in the source code when the widget is first created. Resources set in this manner are said to be *hardcoded* since these resource values take precedence over values in the *resource database* (described in "Setting Resource Values with the Resource Database). Let's see how we can set resource values during widget creation by changing and adding a few resources in `hello.c`.

First we'll change the button layout of our program. Turn to the ControlArea section of the *OLIT Reference Manual* and look at the descriptions of `XtNlayoutType` and `XtNmeasure`. As you can see, the combination `XtNlayoutType` and `XtNmeasure` allows you to change the layout of the child widgets of `control_area`. The current code aligns the buttons horizontally. To align them vertically we set `XtNlayoutType` to `OL_FIXEDCOLS` and `XtNmeasure` to `1`. Open `hello.c` with an editor and add the following lines to the `control_area` argument list:

```
XtNlayoutType,     OL_FIXEDCOLS,
XtNmeasure,        1,
```

Now turn to the OblongButton section of the *OLIT Reference Manual* and look at the description of `XtNlabel`. Let's change our button labels from "Hello!" and "Goodbye!" to "Bon jour!" and "Adios!" We do this by simply changing the string value of `XtNlabel` as follows:

For hello_button:

```
XtNlabel,          "Bon jour!",
```

For goodbye_button:

```
XtNlabel,          "Adios!",
```

After compiling and running, the program should look like the figure below.

Figure 3-1 hello.c with a New Layout and Labels

Setting Color and Font Resources

Most widget resources are set with a simple resource value pair. Color and font resources, however, require a value of type *pixel* or `OlFont`, which needs to be set differently. Instead of a value pair, these resources are set using the following format:

```
XtVaTypedArg,
<resource_name>, XtRString,
"<resource_value>", strlen("<resource_value>")+1,
```

where,
<*resource_name*> is the name of a color or font resource.

<*resource_value*> is the name of a color value listed in `$OPENWINHOME/lib/rgb.txt` or the name of a font listed after executing the `xlsfonts` command.

When XtVaTypedArg is used in a resource specification, the next four arguments are interpreted as special instructions to start a resource converter and set the resource to the result of the conversion. The first argument is the name of the resource to be set. The next argument is the type definition of the next argument, usually XtRString (an Intrinsics-defined string resource type). The third argument is the string value to be converted. The forth argument is the length of the string value, taking into account the string terminator character.

To change the color of the control_area and the goodbye_button, background (assuming you have a color monitor) we look through $OPENWINHOME/lib/rgb.txt and select two colors, say DodgerBlue and bisque. In the control_area argument list we add:

```
XtVaTypedArg,
XtNbackground, XtRString,
"DodgerBlue", strlen("DodgerBlue")+1,
```

In the goodbye_button argument list we add:

```
XtVaTypedArg,
XtNbackground, XtRString,
"bisque", strlen("bisque")+1,
```

To change the font of the hello_button, we type xlsfonts<Ret> in a command window and pick a font name, say bembo-bold, and add the following lines to the hello_button argument list:

```
XtVaTypedArg,
XtNfont, XtRString,
"bembo-bold", strlen("bembo-bold")+1,
```

Modify the hello.c files to see how these changes work.

Getting and Setting Resource Values After Widget Creation

After widgets are instantiated, you can still get and set specific widget attribute values. Let's play with our hello.c program to see how this works.

Currently, when you press the Bon jour! button, "Bon jour!" is displayed, and when you press the Adios! button, "Adios!" is displayed. Let's change the program such that when someone presses either of these buttons, the header in the top level window displays either Adios! or Bon Jour!

To do this, we'll need to write a callback function that retrieves the label resource value of the selected button and assigns this value to the title resource of the top-level application shell widget.

First, we'll call the new callback `buttonCB_label()` and declare it at the beginning of our program:

```
void buttonCB_label();
```

Next we'll attach the callback to both `hello_button` and `goodbye_button` by adding the following line after both widget definitions:

```
XtAddCallback(<widget>, XtNselect, buttonCB_label, toplevel);
```

where `<widget>` is either `hello_button` or `goodbye_button`. `XtNselect` is the callback resource for when an OblongButton is pressed. `buttonCB_label` is the name of the callback function. `toplevel` is the client data we need to send to the callback. Remember that `toplevel` is the widget identifier of the top-level application shell widget. We pass `toplevel` to the callback so we can give it a new title resource value.

Finally we add the callback to the source code to the end of the program:

```
void buttonCB_label(widget, client_data, call_data)
Widget     widget;
XtPointer  client_data, call_data;
{
  String label;
  XtVaGetValues(widget,
                XtNlabel, &label,
                NULL);
  XtVaSetValues(client_data,
                XtNtitle, label,
                NULL);
}
```

In the code segment above, widget is either `hello_button` or `goodbye_button`, `client_data` is `toplevel`, and there is no `call_data` for the OblongButton widget. In the function code, we define a string variable called `label`, then we execute a new Xt function called `XtVaGetValues()`. This function allows you to get the values of a NULL-terminated list of resources from a specified widget, and store them at the addresses specified in the argument list. It has the following format:

```
void XtVaGetValues(object, . . .,NULL)
                   Widget  object
```

where `object` is the widget whose resource values you wish to retrieve, and `. . ., NULL` is a NULL-terminated variable-length list of resource names and addresses where the values of those resources will be stored. In our example, `XtVaGetValues()` gets the value of `XtNlabel` from the pressed button and stores the value in `&label`.

The next function, `XtVaSetValues()`, sets the resource values for a specified widget using a NULL-terminated vararg list. It has the following format:

```
void XtVaSetValues(object, . . .,NULL)
                   Widget  object
```

where `object` is the widget whose resource values you wish to set, and `. . ., NULL` is a NULL-terminated variable-length list of resource/value pairs that override all other resource settings. If `XtVaTypedArg` is used in a resource specification, the next four arguments are interpreted as special instructions to start a resource converter and set the resource to the result of the conversion. Refer to the previous section, "Setting Resource Values at Widget Creation" for details.

In our example, we set the `XtNtitle` resource of `toplevel` to `label`. Note that there is no section in the OLIT Reference Manual about the ApplicationShell widget class. However, since we know that the ApplicationShell widget class is a child of the Shell widget class, we can look at the Shell resources described in Chapter 2 of the OLIT Reference Manual to find out the resources used in ApplicationShell.

Enter the new code into the example, compile, and execute. Now when you press either of the buttons, "Bon jour!" or "Adios!" will appear in the header.

Setting Resource Values with the Resource Database

Resource values not specified in code are retrieved from the *resource database*. When an OLIT program is initialized, a widget resource database is constructed from several system resource files. The database is then embedded in the OLIT program (Figure 3-2).

The resource files are loaded into the database the following order: *applications defaults file*, the *per-user application defaults file*, the *user's defaults file*, an the *user's per-host defaults file*.

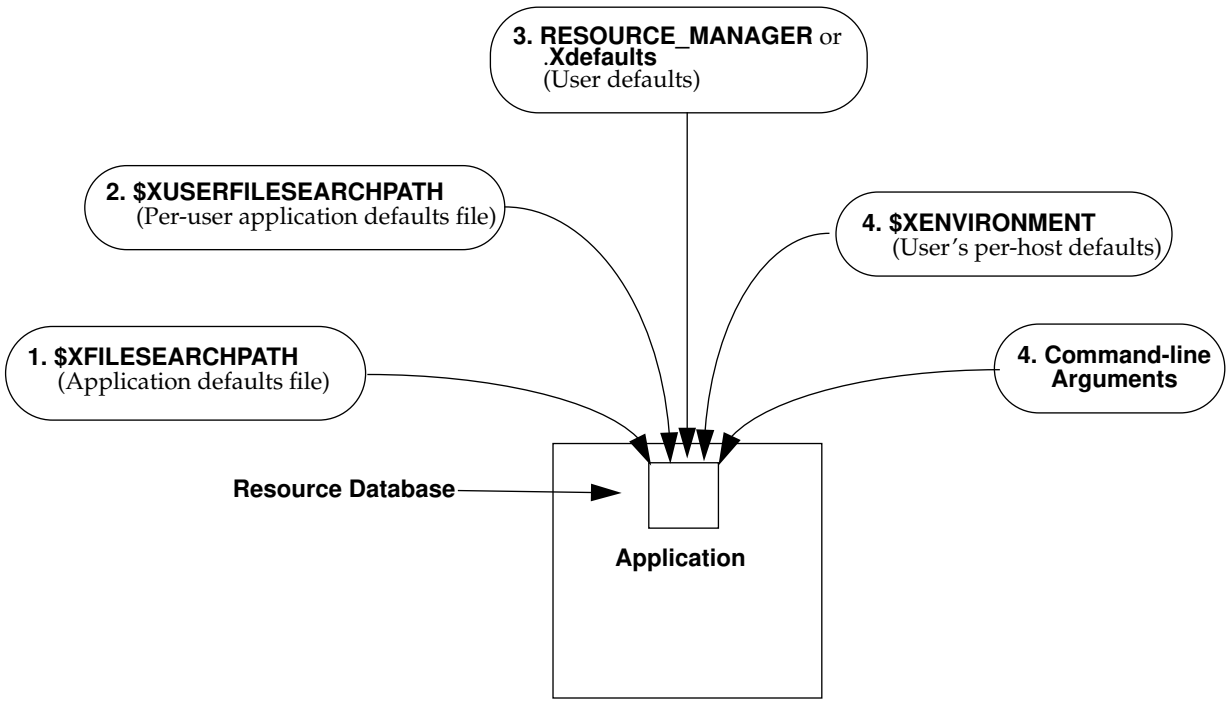

Figure 3-2 Resource Database

The applications defaults file contains the first set of resource values loaded into the resource database. This file is provided by the application writer and supplies the resource values for the application's widgets. The applications defaults file is located in the path identified by environmental variable `XFILESEARCHPATH`. In the Solaris environment this is usually `$OPENWINHOME/lib/locale/app-defaults/<application class name>`.

The per-user application defaults file is loaded into the resource database next. This file allows applications to store application-specific resources specified by the user. The environmental variable `XUSERFILESEARCHPATH` specifies the directory containing per-user application defaults files.

The user defaults file contains user customizations that apply to all applications. The X Toolkit gets the contents of the user defaults file defined by the RESOURCE_MANAGER property on the root window of the display. If this property does not exist, the file .Xdefaults in the user's home directory is used. To look at properties on the root window use xprop command.

The per-host defaults file is last file loaded. It provides user defined attributes for the computer on which the application is running. This file is defined by the XENVIRONMENT environmental variable. If this variable is not defined, the per-host defaults file is .Xdefaults-*host* in the user's home directory, where *host* is the name of the computer on which the application is running.

Each resource file overwrites the resource values of the previous resource file. After all the values of these files are loaded, command line resources are added to the resource database. Refer to *X Window System Programming & Applications with Xt* for details on retrieving resources from the command line, as well as loading the resource database.

You should only set the resource values in program for the resources that you do not want a user to change.

Setting Resources in Resource Files

Resource specifications have the following format:

```
Resource: Value
```

Let's use hello.c and the defaults file Resources_hello to demonstrate how to set resources in default files.

If you haven't already done so, specify ./Resources_hello as the per-host resource file by setting the environmental variable XENVIRONMENT to ./Resources_hello in the window which will run hello. Using the C shell you would enter the following:

```
% setenv XENVIRONMENT ./Resources_hello<Return>
```

Any resource specifications in Resources_hello are added to the command window's resource database. OLIT applications run from this window will inherit these resource specifications. If you execute hello from a different window, it will not inherit the specifications in Resources_hello unless you reset XENVIRONMENT in the new command window.

After setting XENVIRONMENT, edit hello.c and comment out or remove all the resource value pairs in control_area, hello_button, and goodbye_button. When hello is compiled and executed it will look like this:

Figure 3-3 hello.c without Resource Specifications

Note that if a value is not specified for label, hello_button and goodbye_button take their respective widget names as label values—in this case button1 and button2.

Let's change the labels of the application header and the two OblongButtons. Open up the file Resources_hello and add the following lines:

```
hello.title: Greetings
hello.controlarea.button1.label: Hello!
hello.controlarea.button2.label: Goodbye!
```

Now when you run hello you should get the same output as shown in Figure 2-1. The format for setting the resource/value pairs in resource files is:

```
<application>.<parent widgets>.<widget>.<resource>: value
```

<application> is the name of the executable. <parent widgets> are the parent widget names (not identifiers) separated by periods. <widget> is the name of the widget. <resource> is the name of the resource without the XtN prefix, and value is the value of the resource.

A specification does not have to list out all the ancestor widgets on the path to the final widget. A asterisk (*) acts as a wildcard by matching any number of interjacent widgets. For example, run the program after commenting out the two lines we just added in Resources_hello with a ! sign and adding the following line to Resources_hello:

```
*label: Oh no!
```

Any widget that uses the `label` resource will set that resource to “Oh no!” The result is that both buttons say “Oh no!”

Now let’s use the resource file to align the buttons vertically. First comment out `*label: Oh no!` and add the following lines to `Resources_hello`:

```
hello.controlarea.layoutType: fixedcols
hello.controlarea.measure: 1
```

Note that the value for `XtNlayoutType`, `fixedcols`, is different from the value we used in the source code (`OL_FIXEDCOLS`). In many cases the resource value string used in a resource file is different from the value used in code, even though they do the same thing. This difference is documented in the *Values:* category of a resource description in the *OLIT Reference Manual*. The code string is listed first, followed by a slash, and the defaults file string is listed second. `XtNlayoutType` lists the values as follows:

```
OL_FIXEDCOLS/fixedcols
```

Resource values can also be set by widget, resource, or application class. For example, to set the color and font of all OblongButtons, add the following lines to the defaults file and run hello.

```
hello.controlarea.OblongButton.Background: aquamarine
hello.controlarea.OblongButton.font: bembo-bold
```

Instead of specifying a particular widget, we specified the widget class, OblongButton. As the example above shows, you can specify font and color resources by name without going through the converting routine used when setting these resources in code.

Specifying Resources on the Command Line

Resources can be set at the command line by using `-xrm` command line flag. Simply enter the executable followed by `-xrm` and then the resource specification quotes. The resource value pair is then added to the resource database.

You can test this by executing the following command which puts the word “Moose” up in the header.

```
hello -xrm "*title: Moose"
```

Command Line Options

A number of command line options are built into Xt Intrinsics. These are listed in Table 3-1 and include `-bg`, `-fg` (background and foreground color), `-fn`, (font), and `-rv` (set reverse video to on). Using `hello.c` as an example, we can set the background to orange, and the font to courier by starting up the program with the following:

```
hello -fn courier -fg orange
```

Table 3-1 Standard Command Line Options

Option	Resource Name	Effect
-bg or -background	*background	Sets background color
-bd or -bordercolor	*borderColor	Sets border color
-bw or -borderwidth	.borderWidth	Sets border width
-display	.display	Sets display
-fg -foreground	*foreground	Sets foreground color
-fn or -font	*font	Sets font
-geometry	.geometry	Sets size & position of widget.
-iconic	.iconic	Sets resource to “on”
-name	.name	Sets name
-rv or -reverse	.reverseVideo	Sets resource to “on”
+rv	.reverseVideo	Sets resource to “off”
-selectionTimeout	.selectionTimeout	Sets selection timeout
-synchronous	.synchronous	Sets resource to “on”
+synchronous	.synchronous	Sets resource to “off”
-title	.title	Sets title resources
-xnllanguage	.xnllanguage	Sets language
-xrm	not applicable	Allow users to set attribute-value pairs.

Creating New Command Line Options

Additional command line options can be created by passing options table to `XtVaAppInitialize()`. The options table must be an array of type `XrmOptionsDescList()`. Refer to the *X Window System Toolkit* or *The X Window System Programming and Applications with Xt, OPEN LOOK Edition* for details

Dynamic Resource Changing

OLIT also allows some resources to be changed while the program is running. Refer to *The X Window System Programming and Applications with Xt, OPEN LOOK Edition* for details.