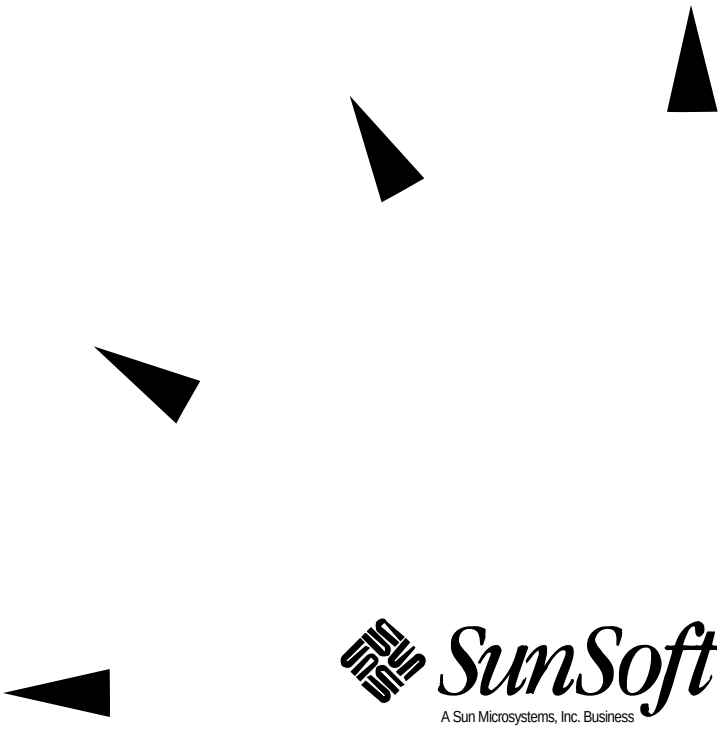
OLIT QuickStart Programmer's Guide



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Please
Recycle

Adobe PostScript

Contents

Figures

Tables

Code Samples

Preface

The purpose of the OPEN LOOK® Intrinsics Toolkit (OLIT) Quick-Start Programming Guide is to teach essential OLIT programming concepts quickly and easily. This manual describes OLIT widget set as well as essential X intrinsics functions. It is not intended as a comprehensive guide OLIT programming or the X Window System. For detailed X Window System® and X Window System Toolkit information, refer to the documents listed at the end of this preface.

Who Should Use This Book

Developers who are considering writing or porting applications to the Solaris operating environment using OLIT should read this book. This book is also helpful to developers who want to quickly assess the power, capability, and programming concepts of OLIT, before committing to its use.

Before You Read This Book

This book presumes that you know C programming and that you are familiar with a computer window system such as OpenWindows or SunView. Knowledge of X is helpful, but not essential.

How This Book Is Organized

Chapter 1, "Basic OLIT Concepts," describes fundamental OLIT concepts.

Chapter 2, "OLIT Program Structure," explains OLIT program structure by examining an example OLIT Program.

Chapter 3, "OLIT Resources," discusses OLIT Resource management.

Chapter 4, "Putting It All Together," shows some OLIT techniques and issues by describing an OLIT program in detail.

"Glossary," describes the terms used in this book.

Related Books

The following books may be of use in learning OLIT. To obtain them, call Sun Express (1-800-873-7869) or visit your local computer bookstore.

- *OLIT Reference Manual*, Sun Microsystems, 1993, P / N 801-5316-10. This book is essential for programming in OLIT.
- *X Window System: Programming and Applications with Xt/ OPEN LOOK Edition*, by Douglas Young & John Pew, published by Prentice Hall, ISBN 0-13-982992-X. The definitive OLIT programming manual.
- *X Window System Toolkit* by Paul Asente & Ralph Swick from Digital Press, 1990, ISBN 1-55558-051-3. The best book on Xt Intrinsics programming.
- *X Toolkit Intrinsics Reference Manual*, O'Reilly & Associates, Inc., ISBN 1-56592-007-4. Lists and describes Xt Intrinsics functions, prototypes, classes, utilities, data types, methods, events, translation tables in a single reference manual.

What Typographic Changes and Symbols Mean

The following table describes the type changes and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	**Meaning**	**Example**
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `Password:`
<AaBbCc123>	Command-line placeholder: replace with a real name or value	To delete a file, type `rm <`*filename*`>.`
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
Code samples are included in boxes and may display the following:		
`%`	UNIX C shell prompt	`system%`
`$`	UNIX Bourne and Korn shell prompt	UNIX Bourne and Korn shell prompt
`#`	Superuser prompt, all shells	Superuser prompt, all shells

Basic OLIT Concepts 1

What is OLIT?

OLIT is an X Window System-based widget set and library used to create applications supporting the *OPEN LOOK* graphical user interface. *Widgets* are user interface objects such as buttons, scrollbars, control areas, text edit areas, and drawing areas. By creating and manipulating desired widgets with the *X Window System Toolkit*, the programmer can create an application user interface. Application code, i.e., the code that performs the actual work on the application's data, is attached to the user interface via callback programs that are executed when the user performs some action on a widget, such as a mouse gesture or keyboard input.

The X Window System Toolkit, also know as the X Toolkit, Xt Intrinsics, or simply Intrinsics, is an X consortium standard library that provides the structure and functions for assembling widgets into a user interface. The X Toolkit provides a few basic widget classes that are usually supplemented with a full-featured widget set such as OLIT. OLIT applications follow an object-oriented, event-driven programming model. For further information about OPEN LOOK, refer to *Open Look Graphical User Interface Guidelines*. For more information on the X Toolkit refer to the *X Window System Toolkit*.

1

The X Window System

Figure 1-1 shows a block diagram of two OLIT programs running in the X Window System. The *X Server* is a program that runs on machines controlling one or more displays. The X Server handles output from an application (called the *client*) to the screen(s), and sends keyboard and mouse input to the appropriate client for processing.

This programming model, called the *client-server computing model*, provides a device independent layer, via the X Server, between applications and the display hardware. Any machine running an X Server, be it a workstation, mainframe, or X terminal, can display X applications regardless of the hardware. In addition to device independence, the distributed architecture of the X Window System allows clients to be run on any machine in a network, and be displayed on any other machine(s) in that network.

The X Server communicates with clients using the *X Window System Protocol* or simply, the X protocol. A C library interface, called *Xlib*, isolates the programmer from the intricacies and detail of the X protocol. Providing another level of abstraction and programming simplicity is the toolkit, which consists of the Xt Intrinsics and the widget set (OLIT). The toolkit provides an object-oriented, event-driven interface via widgets.

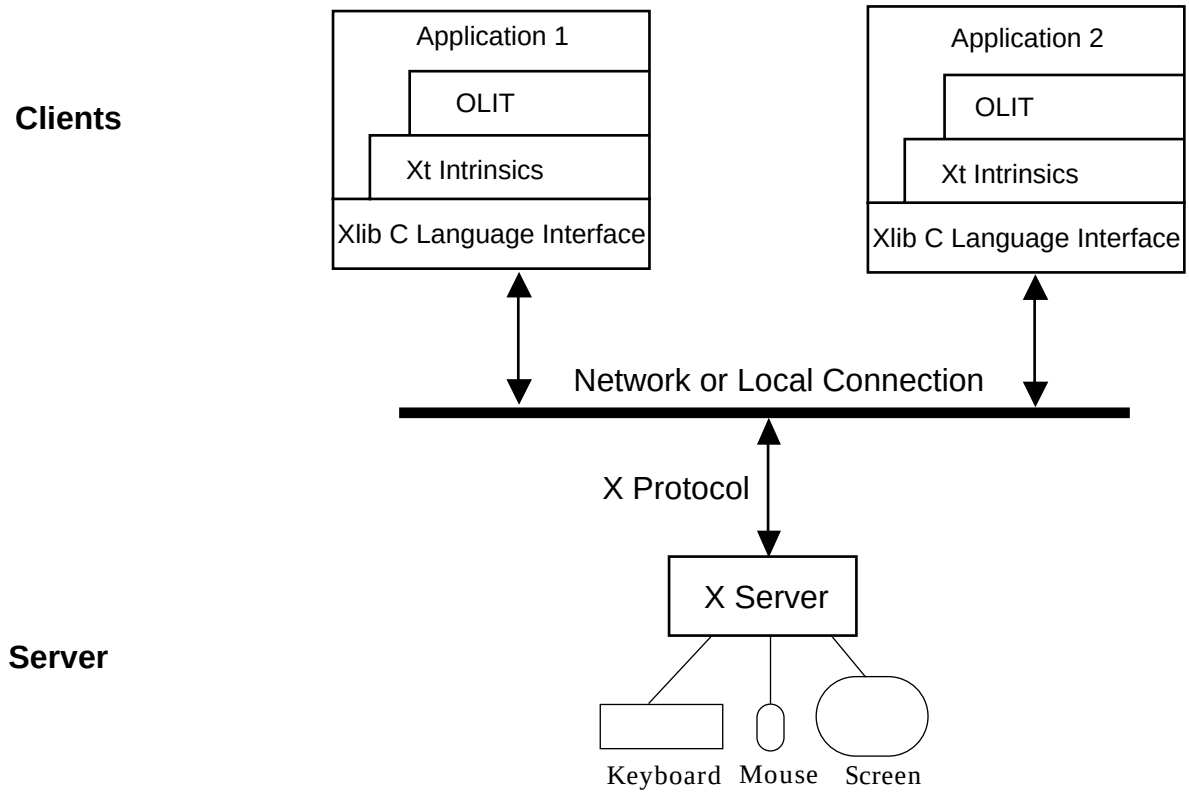


Figure 1-1 OLIT Applications in Client-Server Network

Widgets

Widgets, shown in Figure 1-2, are user interface objects. In the context of the X Window System, widgets are simply specialized X windows that can be created, manipulated, and destroyed. Programmatically, widgets are instantiated data structure types.

Widget types are organized by *class*. For example, there is the ScrollBar widget class, TextEdit widget class, OblongButton widget class, etc. When discussing a widget type, we refer to it as a *widget class*. When discussing a particular widget, we refer to it as a widget *instance*. If we want to create a widget of a particular class, we say we want to *instantiate* a widget.

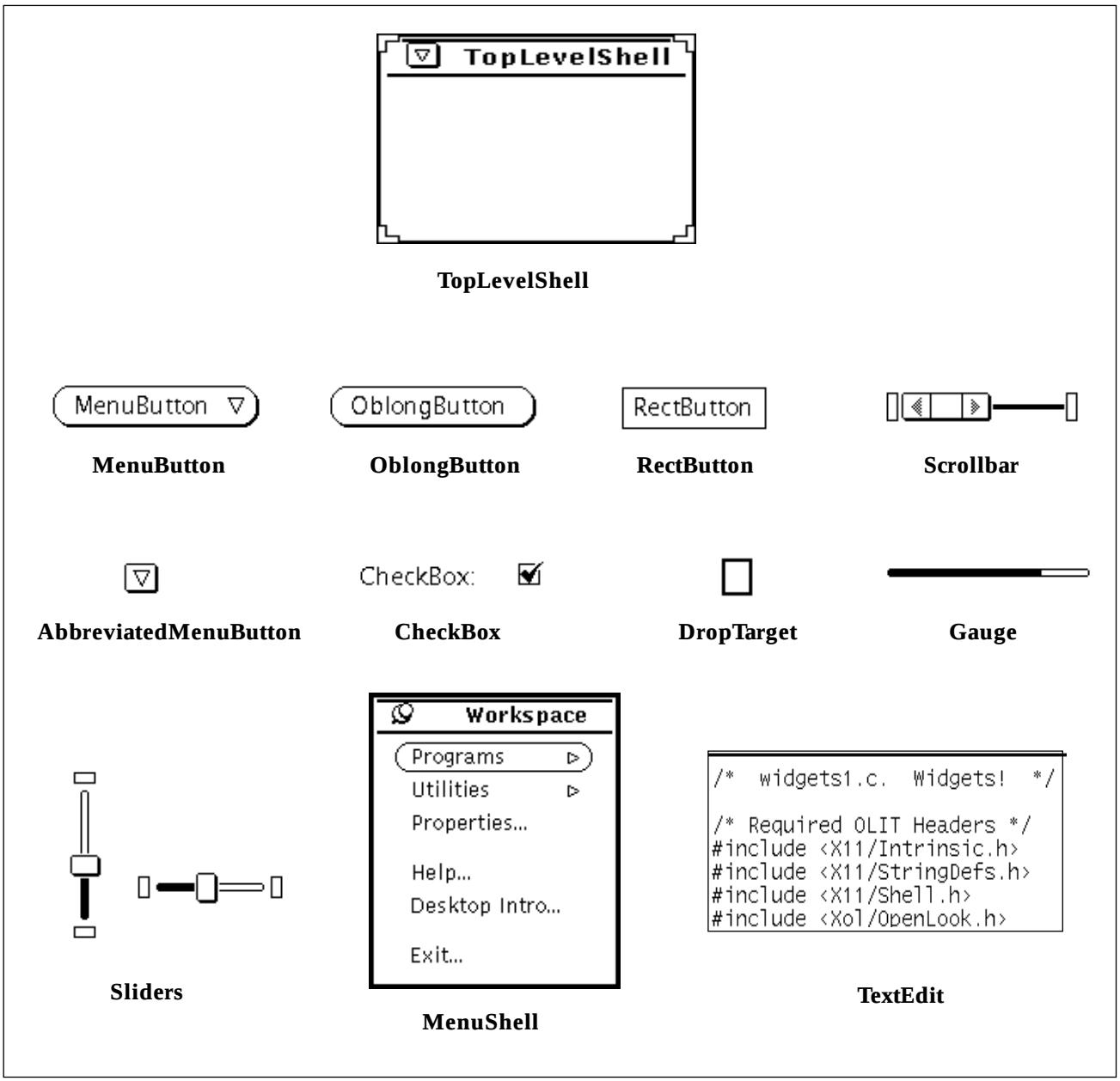


Figure 1-2 Widgets

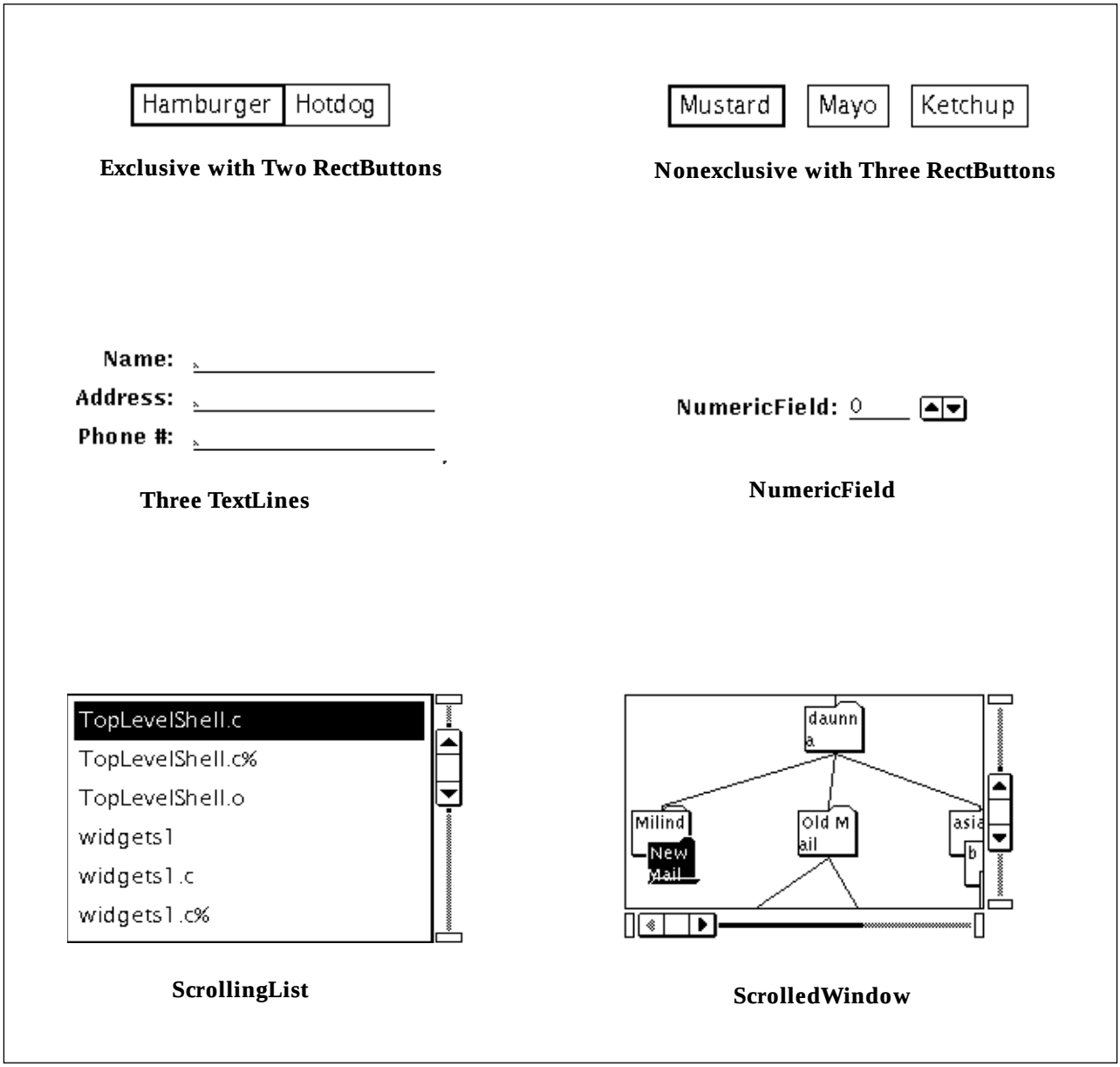


Figure 1-3 Widgets (continued)

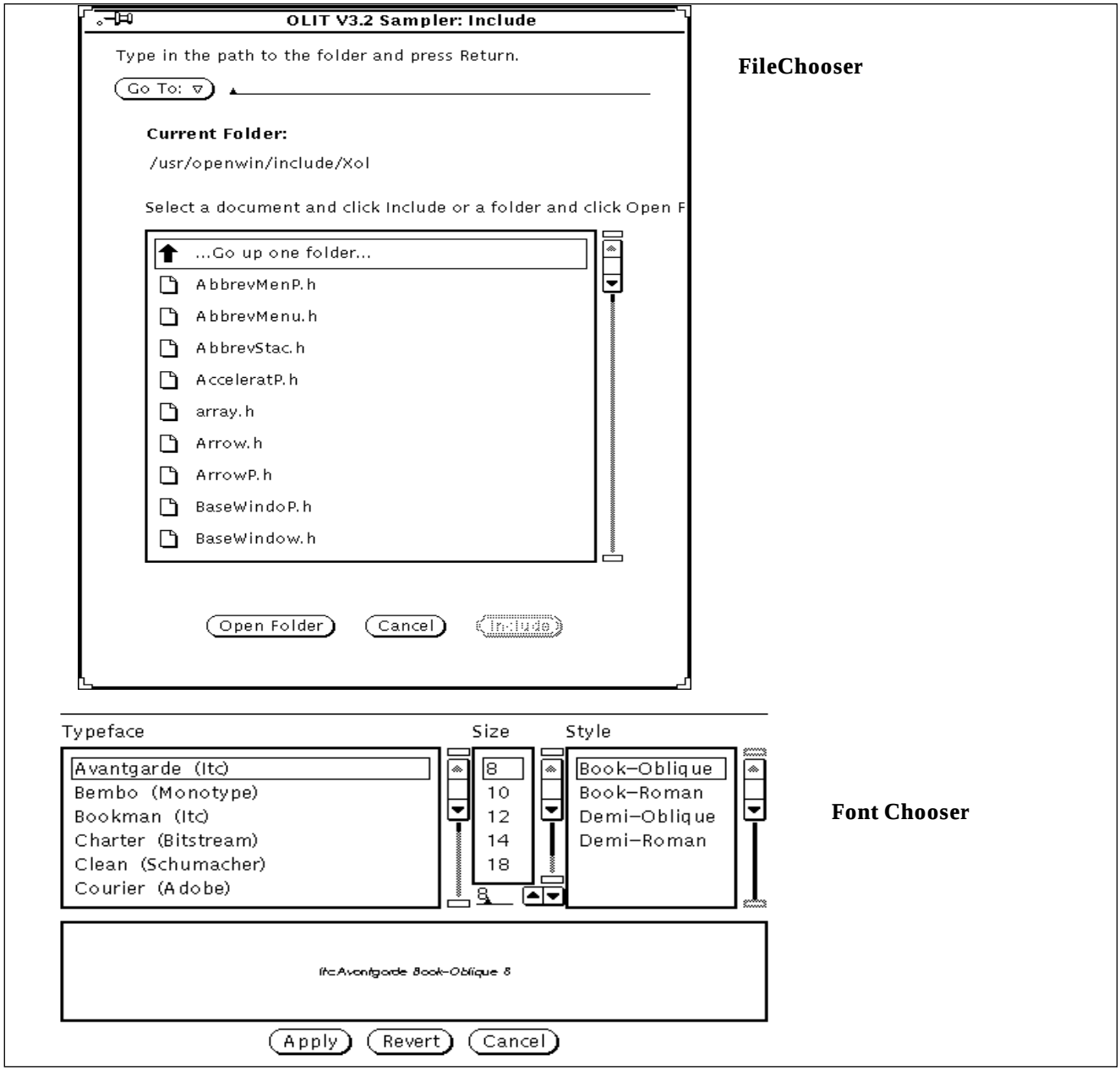


Figure 1-3 Widgets (continued)

John Pew's *X Window System: Programming and Applications with Xt/ OPEN LOOK Edition* categorizes widgets into five groups according to the type of functions they perform. *Action widgets* accept user or application input, and respond to the input with a programmatic action. *Text Control widgets* allow an application to receive and display text or numbers. *Popup widgets* are used to notify the user, bring up a pop-up menu, or prompt for input. *Container widgets* are specialized widgets that manage one or more child widgets. *Manager widgets* are used to combine all widgets into an entire application. A sixth category, called *Function Widgets*, perform specialize functions such as setting application fonts or importing/exporting files.

Table 1-1 OLIT Widgets

Widget Class	**Characteristics and Usage**
Action Widgets	
OblongButton	A button the user can "push" by pressing SELECT on it. When pushed, its border inverts, making it appear "pressed." Used to initiate application-defined actions.
RectButton	A toggle button that executes one set of actions when pressed and another when unpressed. When pressed, the button looks pressed. Button remains pressed until pressed again or unset programmatically.
CheckBox	A toggle that executes one set of actions when checked and another when unchecked. Similar to the RectButton widget.
MenuButton	An oblong button that creates a pop-up menu when user presses MENU on it.
AbbrevMenuButton	Provides a pulldown menu pane (same as the MenuButton widget) user presses MENU on it. Uses less screen space than MenuButton.
Slider	Lets user set a numeric value by sliding drag box along the bar.
Gauge	Graphically displays a read-only numeric value.
Scrollbar	Similar to the slider widget with additional features.
DropTarget	Creates a rectangle to be used as a drop site. Provides visual indication of the progress of a Drag and Drop operation.
Manager Widgets	
BulletinBoard	A composite widget that enforces no particular layout order on its children. Application must specify x- and y-coordinates of each child.
ControlArea	Widget that organizes its children controls in rows and columns.

Table 1-1 OLIT Widgets

Widget Class	Characteristics and Usage
DrawArea	Provides a window on which to render images using Xlib calls.
RubberTile	Organizes its children vertically or horizontally in a single file. Assigns relative weights to each child so that it expands or contracts a certain percentage of size changes of the RubberTile.
Form	Organizes children relative to the position of other children. Manipulates these layout when Form is resized, new children are added, moved, resized, unmanaged, remanaged, rearranged, or destroyed.
Text Widgets	
FooterPanel	Attaching a footer message to the bottom of a base window.
NumericField	Provides a one-line input field for alphanumeric text.
StaticText	Displays an uneditable block of text.
TextEdit	Provides a multi-line text editing window.
TextLine	One-line input field for text data. Replaces TextField.
Container Widgets	
Caption	Provides a convenient way to label a widget.
Exclusives	Manages a set of rectangular buttons to create a one-of-many button selection object
Nonexclusives	Manages a set of rectangular buttons or check boxes to create a several-of-many button selection object.
FlatCheckBox	Same functionality as a `NonExclusives` widget managing CheckBox widgets, but instead of creating individual CheckBoxes as children of a container widget, it creates sub-objects that have the same behavior as the CheckBoxes. Improves performance since fewer widgets are created.
FlatExclusives	Same functionality as an `Exclusives` widget managing RectButtons, as the same advantages as described in FlatCheckBox.
FlatNonexclusives	Provides the same functionality as a `Nonexclusives` widget managing RectButtons. Has the same advantages as described in FlatCheckBox.
ScrolledWindow	Provides a scrolled window view of a text or graphics pane.
ScrollingList	Provides a list of items the user can scroll through and select. Items can be exclusive or multiple choice non-exclusive. Users can edit items.

Table 1-1 OLIT Widgets

Widget Class	Characteristics and Usage
Popup Widgets	
NoticeShell	Creates a pop-up user notification message. Warns user of a problem; allows user to confirm a choice. Interaction with the application stops until the `NoticeShell` is popped down.
MenuShell	Creates a pop-up menu not associated with `MenuButton` or `AbbrevMenuButton`. Has the same features as a `MenuButton` widget (except those related to menu creation).
PopupWindowShell	Creates a pop-up property window allowing users to set properties of an application.

By creating and manipulating the desired widgets, the developer assembles the application's user interface. After the user interface is assembled, the application code, i.e., code that performs computations on the application's data, is attached to the desired widget as *callback procedures*, or simply *callbacks*. A callback procedure is a operation that gets executed when the widget receives some event, such as a mouse selection, a scrollbar being moved, or a text field being entered. Attaching specific procedures to specific widgets allows you to produce modular source code.

Widget Resources

A widget resource is a named piece of data that sets a specific widget attribute. Resources set attributes such as the color, font, layout, alignment, label, or callback list of a widget. A widget's resource values are usually set in a resource file for flexibility, but can also be set in the application code during widget creation or as a command line option. Widget resources are specified in

resource/value pairs with the resource name followed by the resource value (see Code Example 1-1). The OLIT Reference Manual lists and describes all the widget class resources.

Resource/Value Pair in Application Code

```
hello_button = XtVaCreateManagedWidget( “button1”,
                    oblongButtonWidgetClass, control_area,
                    XtNlabel, “Hello!”,
                    NULL);
```

Resource/Value Pair in a Resource File

```
hello.controlarea.button1.label: Hello!
hello.controlarea.OblongButton.background: aquamarine
```

Code Example 1-1 OLIT Resource / Value Pairs (in **bold**)

Widget Class Hierarchy

The X Toolkit organizes widgets by class. Class defines the characteristics, operations, and resources of a widget type. This includes the resources a widget class uses. All widget classes are *subclassed* from other widget classes. That is, a widget class is created by modifying and specializing another widget class called its *superclass*. The subclass inherits some or all of the characteristics of it’s superclass. Class architecture and inheritance make it easier to create new widgets because subclasses use much of the same code and declarations as its superclass.

From an programmer’s standpoint, the OLIT widget class hierarchy and architecture would seem of minimal interest unless you were going to create a new widget class by subclassing another widget. Understanding a little about the architecture, however, will help understand the relationship and shared characteristics of between all widgets.

Figure 1-3 shows the OLIT widget class hierarchy. Note that Xt Intrinsics defines a number of widget classes that are superclasses of all other widget classes. For example, all widget classes are subclassed from the Core widget class, therefore, all widgets inherit the Core widget resources.

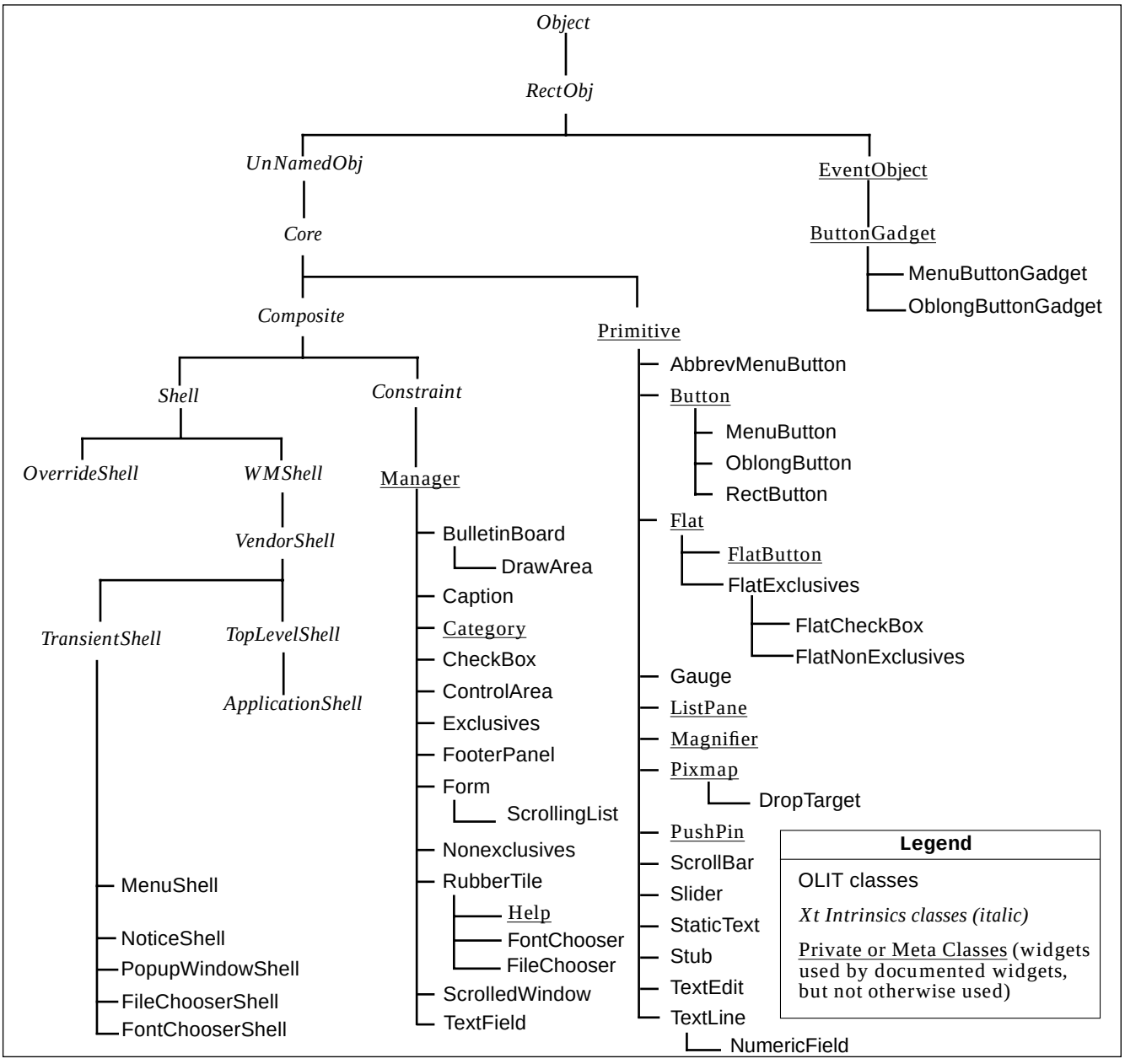


Figure 1-3 OLIT Class Hierarchy Tree

1

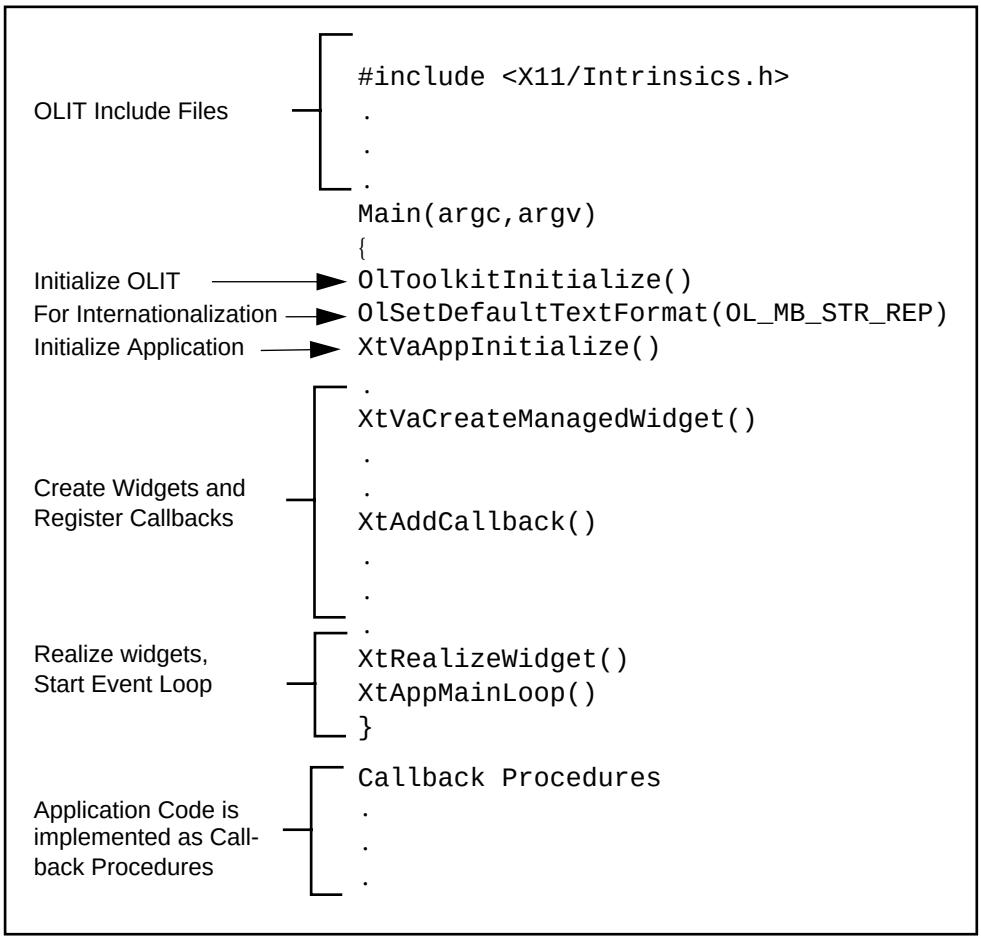
OLIT Program Structure 2

Overview

The best way to learn about the OLIT program structure is to look at an OLIT program. The example program we use throughout this chapter, `hello.c` (Code Example 2-2 on page 16), contains all the major elements of an OLIT program. These elements are:

- OLIT and Xt Intrinsics include files.
- Initialization steps consisting of calls to `OlToolkitInitialize()`, `XtAppInitialize()`, and `XtVaAppInitialize()`.
- Creation of widgets and attaching of callbacks to the widgets.
- A call to `XtRealizeWidget()` to make the widgets appear on the screen.
- A call to `XtAppMainLoop()`, an event gathering loop that gets X events from the display connection and sends them to the appropriate widgets for processing.
- The application code, which is implemented as callback procedures.

`hello.c`, like all OLIT applications are structured as follows:[1]



1. The code line `OlSetDefaultTextFormat(OL_MB_STR_REP)` will only work in OpenWindows 3.2 or Asian OpenWindows 3.1. For details on how to internationalize an OLIT application refer to the OLIT Reference Manual.

Program Structure Example—hello.c

You may obtain `hello.c`, its helper files `Makefile` and `Resources_hello`, as well as the another example program used in the manual by sending e-mail to greg.kimura@Eng.sun.com with the subject line “OLIT Quick-Start Programs”. To compile and execute `hello.c`, copy `hello.c`, `Makefile`, and `Resources_hello` into a work directory. Set the environmental variable `XENVIRONMENT` to `./Resources_hello` and enter `make<return>`. `Makefile` (Code Example 2-1) compiles the program. To run the program, type `hello<return>`.

Code Example 2-1 Makefile

```
INCLUDE   =  -I${OPENWINHOME}/include
CFLAGS    =  ${INCLUDE} -g
LIBS      =  -L${OPENWINHOME}/lib -lXol -lXt -lX11

TARGETS = hello


all: ${TARGETS}

${TARGETS}: $$@.c $$@.o
    ${CC} ${CFLAGS} ${LDFLAGS} -o $@ $@.o ${LIBS}

clean:
    rm -f core ${TARGETS} *.o
```

`hello.c` (Figure 2-1) consists of four widgets: a top-level Shell widget containing the user interface, a ControlArea widget containing two OblongButtons, and the two OblongButton widgets themselves. A callback procedure is attached to the Hello and Goodbye buttons such that selecting the buttons with the mouse will print out the word Hello! or Goodbye!

Figure 2-1 `hello.c` Compiled and Executed

Code Example 2-2 `hello.c` Source Code

```
        /* Required OLIT Headers */
1   #include <X11/Intrinsic.h>
2   #include <X11/StringDefs.h>
3   #include <Xol/OpenLook.h>

        /* Specific Widget Headers */
4   #include <Xol/ControlAre.h>
5   #include <Xol/OblongButt.h>

6   main(argc,argv)
7       int             argc;
8       char            *argv[];
9   {
10      static char     *data_h = "Hello!";
11      static char     *data_g = "Goodbye!";
12      XtAppContext    app;
13      Widget          toplevel, control_area, hello_button, goodbye_button;

14      void ButtonCB();

        /* Initialize toolkit, set strings to multibyte, create the toplevel shell.          */
        / * Note that the OlSetDefaultTextFormat line is for internationalization and      */
        / * will only work with OpenWindows 3.2 or Asian OpenWindows 3.1. If you are*/
        / * not using one of these versions, comment out this line.                        */
15      OlToolkitInitialize((XtPointer)NULL);
16      OlSetDefaultTextFormat(OL_MB_STR_REP);
17      toplevel = XtVaAppInitialize(&app, "Hello", (XrmOptionDescList)NULL,
18                          0, &argc, argv, (String *) NULL,
19                          NULL);

        /* create top manager widget */
20      control_area = XtVaCreateManagedWidget( "controlarea",
21                          controlAreaWidgetClass, toplevel,
22                          XtNhSpace,    10,
23                          XtNvSpace,    10,
24                          XtNvPad,      10,
25                          XtNhPad,      10,
26                          NULL);

        /* create hello button widget */
27      hello_button = XtVaCreateManagedWidget( "button1",
28                          oblongButtonWidgetClass, control_area,
29                          XtNlabel, "Hello!",
30                          NULL);

        /* identify a callback procedure for hello_button */
31      XtAddCallback(hello_button, XtNselect, ButtonCB, (XtPointer)data_h);

        /* create goodbye button widget */
32      goodbye_button = XtVaCreateManagedWidget( "button2",
```

Code Example 2-2 `hello.c` Source Code

```
33                          oblongButtonWidgetClass, control_area,
34                          XtNlabel, "Goodbye!",
35                          NULL);

          /* identify a callback procedure for goodbye_button */
36      XtAddCallback(goodbye_button, XtNselect, ButtonCB, (XtPointer)data_g);

          /* realize the widgets and begin the main loop */
37      XtRealizeWidget(toplevel);
38      XtAppMainLoop(app);
39  }

          /*  procedure called when hello_button selected  */
40  void ButtonCB(widget, clientdata, calldata)
41      Widget          widget;
42      XtPointer       clientdata;
43      XtPointer       calldata;
44  {
45      printf("%s\n", (char *)clientdata);
46  }
```

OLIT Include Files

Let's examine the source code for *`hello.c`* in detail.

```
1   #include <X11/Intrinsic.h>
2   #include <X11/StringDefs.h>
3   #include <Xol/OpenLook.h>
```

Lines 1 through 3 are the header files required for all OLIT programs.

```
4   #include <Xol/ControlAre.h>
5   #include <Xol/OblongButt.h>
```

Lines 4 and 5 are header files for each class of widget created in the program. If another widget is added to the program, for example, a scrollbar, we would have to include the scrollbar widget class file:

```
#include <Xol/Scrollbar.h>
```

Widget class header files are listed under the *Synopsis* section of the desired widget in the *OLIT Reference Manual*.

Declarations

```
 6   main(argc,argv)
 7       int             argc;
 8       char            *argv[];
 9   {
```

Lines 6 through 9 start the main body of the program.

```
10       static char    *data_h = "Hello!";
11       static char    *data_g = "Goodbye!";
```

Line 10 and 11 create two static character string variables called `data_h` and `data_g` and initialize them with the strings "Hello!" and "Goodbye!" These strings will be used by the callback procedure described later.

```
12       XtAppContext   app;
```

Line 12 defines an *application context* of type `XtAppContext`. An application context is a pointer to an opaque data structure containing all the information the toolkit maintains for one application. This line must appear in all OLIT programs. `XtAppContext` is an Intrinsics data type. There are a number of Intrinsics data types that are used in OLIT. In this guide, however, we'll describe only the ones used. For a complete listing of the X Toolkit data types, refer to the *X Toolkit Intrinsics Reference Manual* or the *X Window System Toolkit*.

```
13      Widget          toplevel, control_area, hello_button, goodbye_button;
```

Line 13 defines four variables of type `Widget` (another Intrinsics data type) called `toplevel`, `controlarea`, `hello_button`, and `goodbye_button`.

```
14       void ButtonCB();
```

Line 14 specifies a callback procedure called `ButtonCB` that returns no data. This callback will be attached to the button widgets, which we will discuss shortly.

Initialization and Creating the Top-level Shell Widget

```
15      OlToolkitInitialize((XtPointer)NULL);
```

Line 15 initializes OLIT. The program must be initialized before any widgets are created or OLIT routines are used. `OlToolkitInitialize()` is passed a null argument of type `XtPointer` (an X type), which is reserved for future use. Note that the prefix `Ol` usually signals that a function is an OLIT function. Intrinsics functions are prefaced with `Xt`.

```
16      toplevel = XtVaAppInitialize(&app, "Hello", (XrmOptionDescList)NULL,
17                              0, &argc, argv, (String *) NULL,
18                              NULL);
```

Lines 16 through 18 use the Intrinsics function `XtVaAppInitialize()` to initialize the application context, parse the command line, set shell widget resource values, and create the initial application shell widget. The shell widget is returned as an opaque pointer and assigned to the variable `toplevel`. The widget variable is referred to as a widget *handle* or *identifier*.

The argument `&app` is the address of the application context.

`"Hello"` is the application class name. Intrinsics uses the class name to specify attribute resources for application classes. Note that the name is different from the widget identifier, i.e., `toplevel`.

The next two arguments, `(XrmOptionDescList)NULL` and `0` specify the command line options table and the number of option entries. In our example, there are no command line options.

`&argv` and `argc` are a pointer and count of command line options being passed into the application.

The next options are *fallback resources*. Fallback resources are used if no other application default files are provided (see Chapter 3, "OLIT Resources"). Fallback resources are specified as type `String *`. Since there are no fallback resources in this example, we terminate with `(String *)NULL`.

The final arguments to `XtVaAppInitialize()` are the shell widget's `NULL`-terminated resource/value pairs. Generally, you should specify resource values in resource files rather than in application code because it allows the

user to easily change the resource value if he doesn't like your value. For internationalized applications, the ability to easily change resource values is crucial.

In this case, we don't specify any resource/value pairs. If we did want to add shell widget resource/value pairs, we would have to include `<X11/Shell.h>` to our list of include files. Widget resources can be set by a number of methods. These methods are described in Chapter 3, "OLIT Resources."

For details on `XtVaAppInitialize()` or any Xt Intrinsics function, refer to the *X Toolkit Intrinsics Reference Manual* or *X Window System Toolkit*.

Creating Widgets

```
19     control_area = XtVaCreateManagedWidget( "controlarea",
20                          controlAreaWidgetClass, toplevel,
21                          XtNhSpace,    10,
22                          XtNvSpace,    10,
23                          XtNvPad,      10,
24                          XtNhPad,      10,
25                          NULL);
```

Lines 19 to 25 create a ControlArea widget and assign it to the identifier `control_area`. The ControlArea widget is called a Manager widget because it doesn't display any information, but manages other widgets. In this case, the ControlArea widget manages two OblongButtons.

The Xt intrinsics call `XtVaCreateManagedWidget()` is one of the most commonly used functions in OLIT. `XtVaCreateManagedWidget()` has the following form:

XtVaCreateManagedWidget(name, widget_class, parent, args, NULL)

The first argument, `name`, is a programmer-defined name for the widget. Note that widgets have both a name and identifier. Essentially, widget names are used to specify resource names in resource files and widget identifiers are used in OLIT and Intrinsics functions. The name we have given for this ControlArea widget is `"controlarea"`.

The second argument, `widget_class`, is a widget class pointer that specifies the class of widget created. The widget class pointer is listed in the *OLIT Reference Manual* in the section pertaining to the desired widget. Usually the

widget class can be derived by adding the word `WidgetClass` to the end of the widget class name and making the first letter lowercase. In this example the widget class pointer is `controlAreaWidgetClass`

The third argument, `parent`, specifies the *parent* widget of the widget being created. Except for shell widgets, all widgets are *children* of some parent widget. An application organizes widgets into a hierarchy of parents and children, the root of which is the shell widget created by `XtVaAppInitialize()`. The widget tree for our example program is shown below.

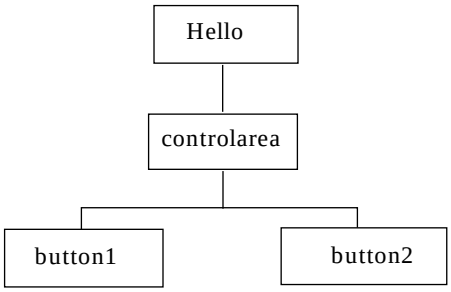


Figure 2-2 `hello.c` Widget Instance Tree

Note that the widget application hierarchy (also known as the widget instance hierarchy) is different from the widget class hierarchy described in “Widget Class Hierarchy” on page 10. The class hierarchy describes the fundamental OLIT widget architecture. Application hierarchy describes the parentage of an application’s widgets.

`args`, the final arguments passed to `XtVaCreateManagedWidget()`, are the NULL-terminated resource/value pairs. Again, it is usually better to set resource values in resource files rather than setting them during widget creation. However, we do hardcode a few resource values in this example for demonstration purposes. In Chapter 3, “OLIT Resources,” we describe how to set OLIT resources in resource files.

Four resource are set in our example. These are `XtNhSpace`, `XtNvSpace`, `XtNvPad`, and `XtNhPad`. Open the *OLIT Reference Manual* to the ControlArea section in Chapter 3. After the ControlArea resource table is a description of the resources specific to the ControlArea. These resources define the horizontal and vertical space between `controlarea`’s children (buttons), as well as the

space from the edge of `controlarea` to the edge of its children. Note that you do not have to define all resources. OLIT assigns default values if none are specified.

```
26     hello_button = XtVaCreateManagedWidget( “button1”,
27                          oblongButtonWidgetClass, control_area,
28                          XtNlabel, “Hello!”,
29                          NULL);
```

Lines 26 through 29 create an OblongButton instance with the word “Hello!” on it and assign it to the identifier `hello_button`. Argument 1, widget name, is `button1`. Argument 2, widget class pointer, is `oblongButtonWidgetClass`. Argument 3, widget parent, is `control_area`. One resource value argument, `XtNlabel`, is specified as `Hello`!

```
30     XtAddCallback(hello_button, XtNselect, ButtonCB1, (XtPointer)data_h);
```

Line 30 uses the Intrinsics function `XtAddCallback()` to attach a callback procedure executed when the hello button is selected. When mouse selected, the callback procedure prints “Hello!”

Attaching Callbacks

`XtAddCallback()` adds callbacks to a widget action. It uses the following form:

XtAddCallback(widget, callback_resource, callback, client_data)

`widget` specifies the widget identifier to which the callback is attached, `hello_button`, in our example.

`callback_resource` is the widget resource that specifies the *callback list* to which the callback procedure is to be added. Callback resources are very different from other widget resources. Unlike other widget resources, callback resources are not defined as resource/value pairs in null-terminated lists or in resource files. Callback resources are specified as a single resource name in `XtAddCallback()` to attach a callback procedure that is executed when a specific user action is performed on the widget.

The reason for using resources to specify callbacks is that a widget may respond differently to different user actions. For example, a CheckBox widget (below) allows a user to toggle between two states—checked and unchecked. You can invoke one callback when the box is checked and another callback when it is unchecked by setting `callback_resource` to `XtNselect` for one callback and setting `callback_resource` to `XtNunselect` for another.

You can also attach more than one callback to a specific widget action by calling `XtAddCallback()` repeatedly and setting the `callback` argument to a different callback each time. The callbacks will be called in the order they were added to the callback list.

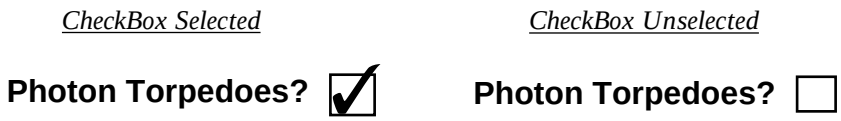


Figure 2-3 CheckBox Widget

Returning to the program example, we want to execute a single operation when the user selects the hello button. The OblongButton section of the *OLIT Reference Manual*, describes `XtNselect` of class XtCCallback, and that it specifies “the list of callbacks invoked when OblongButton is selected.” Thus, `callback_resource` is specified as `XtNSelect`.

The next argument in `XtAddCallback()`,`callback`, gives the name of the callback procedure. In our example this is `ButtonCB`.

The last argument, `client_data`, is data passed to `callback` when `callback` is invoked; this is application-specific data to be used within the callback procedure. `client_data` must be of type `XtPointer`. In this example, `client_data` is the variable `data_h` with the contents “`Hello!`” which the application uses as a string to print out when the button is selected.

```
31     goodbye_button = XtVaCreateManagedWidget( “button2”,
32                          oblongButtonWidgetClass, control_area,
33                          XtNlabel, “Goodbye!”,
34                          NULL);
```

Lines 31 through 34 create another OblongButton with the label “Goodbye!” on it and assign it to the identifier `goodbye_button`. The widget name is `button2`. The widget class pointer is `oblongButtonClassWidget`. The parent widget is `control_area`. Only one resource value argument, `XtNlabel`, is specified. Its value is `Goodbye!`

```
35      XtAddCallback(goodbye_button, XtNselect, ButtonCB, (XtPointer)data_g);
```

Line 35 attaches the same callback, `ButtonCB`, to be executed when the Goodbye button is selected. The pointer `*data_g` with the contents Goodbye! is passed to the `ButtonCB()` callback as client data.

Realizing Widgets and Starting Event Loop

```
36      XtRealizeWidget(toplevel);
37      XtAppMainLoop(app);
38  }
```

`XtRealizeWidget` makes all the widgets appear on the screen and `XtAppMainLoop` gathers input events from the X server and distributes them to widgets. This routine continues looping until the program is exited.

hello.c Callback Procedure

```
39  void ButtonCB(widget, clientdata, calldata)
40      Widget          widget;
41      XtPointer       clientdata;
42      XtPointer       calldata;
43  {
44      printf(“%s\n”, (char *)clientdata);
45  }
```

Lines 39 to 45 define the callback invoked when the Hello! or Goodbye! button are selected. Callback procedures have the following prototype:

void Callback(widget, client_data, call_data)

`widget` is the widget identifier to which the callback is registered. In our example this is `hello_button` or `goodbye_button` depending on which button was pressed. The second argument, `client_data`, is data passed by

the application in the call to `XtAddCallback`. In our example, this is `(XtPointer)data_h` (`Hello!`) or `(XtPointer)data_g` (Goodbye!). `call_data` is data passed by the widget. The type and purpose of this data (if any) is described under the specific widget description in the *OLIT Reference Manual*. OblongButton passes no widget data to its callbacks. Line 44 is the callback procedure itself, which prints the expression “Hello!” or “Goodbye!”

2

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*.

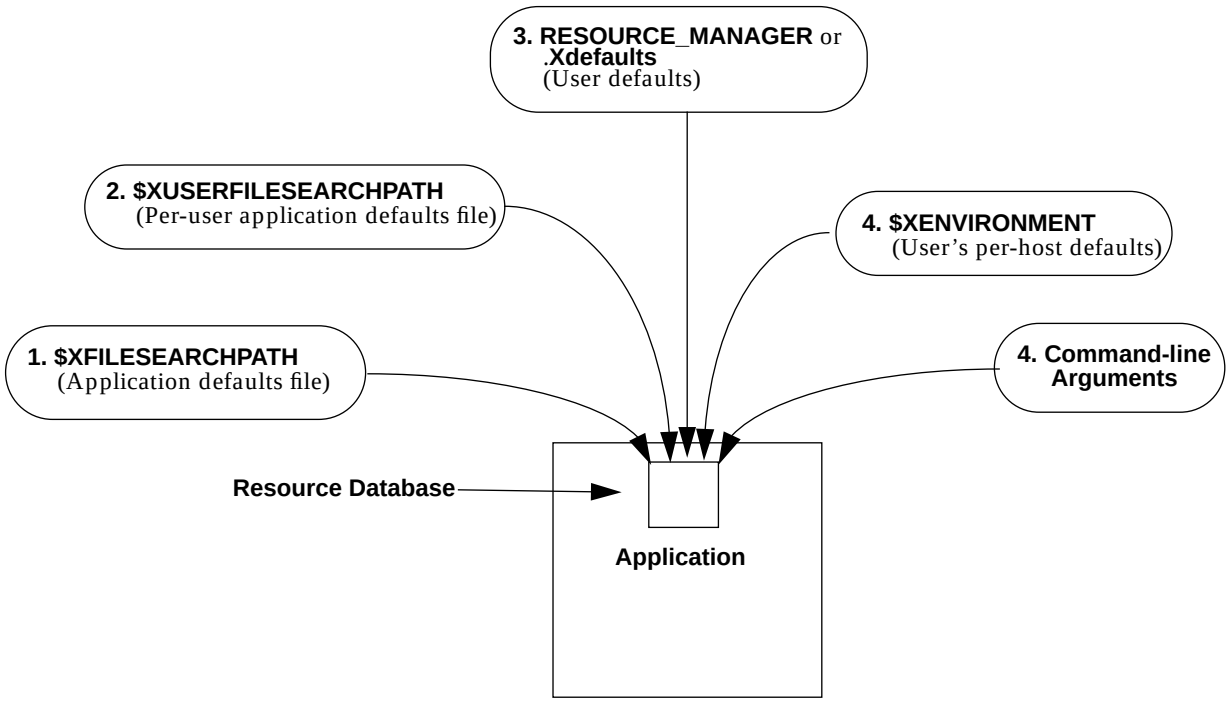


Figure 3-2 Resource Database

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.

Putting It All Together 4

Much of OLIT programming is figuring out how the various widgets work, and how to assemble them into a user interface. The best way to teach this is to show it in a real OLIT program. The program described in this chapter, `Translator.c`, translates a some common English expressions into several foreign languages. Examine the run-time application and the commented source code to get an idea of some of the issues involved in creating an OLIT application.

You may obtain `Translator.c` and the its helper files (`Makefile`, and `Resources_translator`) by sending mail to greg.kimura@Eng.sun.com with the subject line “OLIT Quick-Start Programs”. To compile and execute `Translator.c`, copy `Translator.c`, `Makefile`, and

`Resources_translator` into a work directory. Set the environmental variable `XENVIRONMENT` to `./Resources_translator`. Enter `make<return>` to make. Enter `Translator<return>` to run.[1]

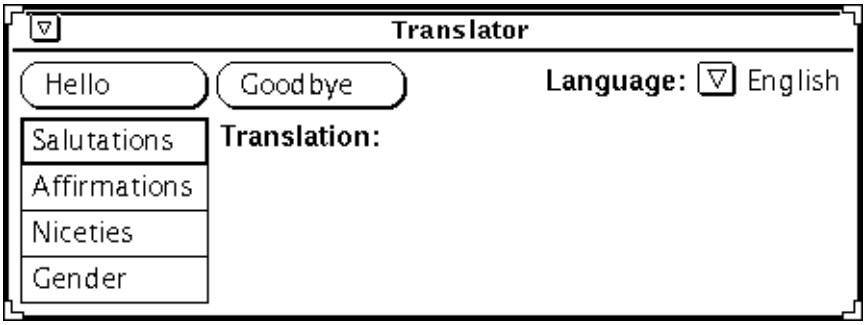


Figure 4-1 Translator

Translator translates some common English words into several different languages. Here's how to use it:

1. Selected a language by pulling down the abbreviated menu button and highlighting the desired language. The language will be displayed next to the abbreviated menu button.
2. Select the type of words you wish to translate by pressing the Salutations, Affirmations, Niceties, or Gender buttons.
3. Press one of the Oblong buttons for a translation of the desired word. The translated word will appear next to the label Translation:.

Table 4-1 lists the variables in `Translator.c`. Code Example 4-1 shows the resource file for `Translator.c`. Code Example 4-2 lists the source code.

1. Note that if you are not using OpenWindows 3.2 or Asian OpenWindows 3.1, you may have trouble compiling this program unless you remove the line which reads OlToolkitInitialize((XtPointer)NULL);

Table 4-1 Translator.c Variable List

Variable	Type	Description
abbrevmb	AbbrevMenuButton	Pulldown menu button for the Language buttons.
answer	StaticText	Foreign language translation of pressed button.
control_area	ControlArea	Manager widget.
goodbye_button	OblongButton	Right word button.
hello_button	OblongButton	Left word button.
holder	Exclusives	An Exclusives widget used to hold the RectButtons.
i	int	Counter.
israel[]	char	Israeli language button labels. Yiddish and Hebrew.
israeli_buttons	OblongButtons	Yiddish and Hebrew selection button widgets.
israeli_menu	MenuButton	Israeli MenuButton widget.
isrl_menu	MenuShell	,MenuShell subwidget of israeli_menu.
lang_button_cb()	void	Callback procedure called when a language (lang_button) is selected. Puts the name of the language next to the AbbreviatedMenuButton.
lang_buttons[]	char	Labels for the language selection buttons. English, Spanish, Japanese, German, French, Swahili
language_buttons[]	OblongButtons	English, Spanish, Japanese, German, French, Swahili selection button widgets.
lang_label	StaticText	Language: label to the left of the abbreviated menu button.
lang_menu	MenuShell	MenuShell subwidget of abbrevmb.
new_header_cb()	void	Callback procedure called when the hello and goodbye button are selected. Sets application header to the string in the Hello / Goodbye buttons.
opposites[]	char	Names of the word pairs. Salutations, Affirmations, Niceties, Gender.
opposites_cb()	void	Callback procedure called when an opposites button (Salutations, Affirmations, Niceties, and Gender) is selected. When button is selected, the corresponding word pair is displayed in the Hello and Goodbye buttons.

Table 4-1 Translator.c Variable List

Variable	Type	Description
Opposites[]	RectButtons	Opposites selection buttons. Salutations, Affirmations, Niceties, Gender.
opposites[]	char	Opposites selection button labels. Salutations, Affirmations, Niceties, Gender.
preview_lang	StaticText	Previews the selected language.
spacer_button	OblongButton	Used to create a blank space between the Goodbye button and “Language”.
toplevel	TopLevelShell	Top level shell widget.
translation	StaticText	Static label. “Translations:”
translation_cb()	void	Callback executed when a word button is pressed. Matches word and language. Prints out the foreign language translation.

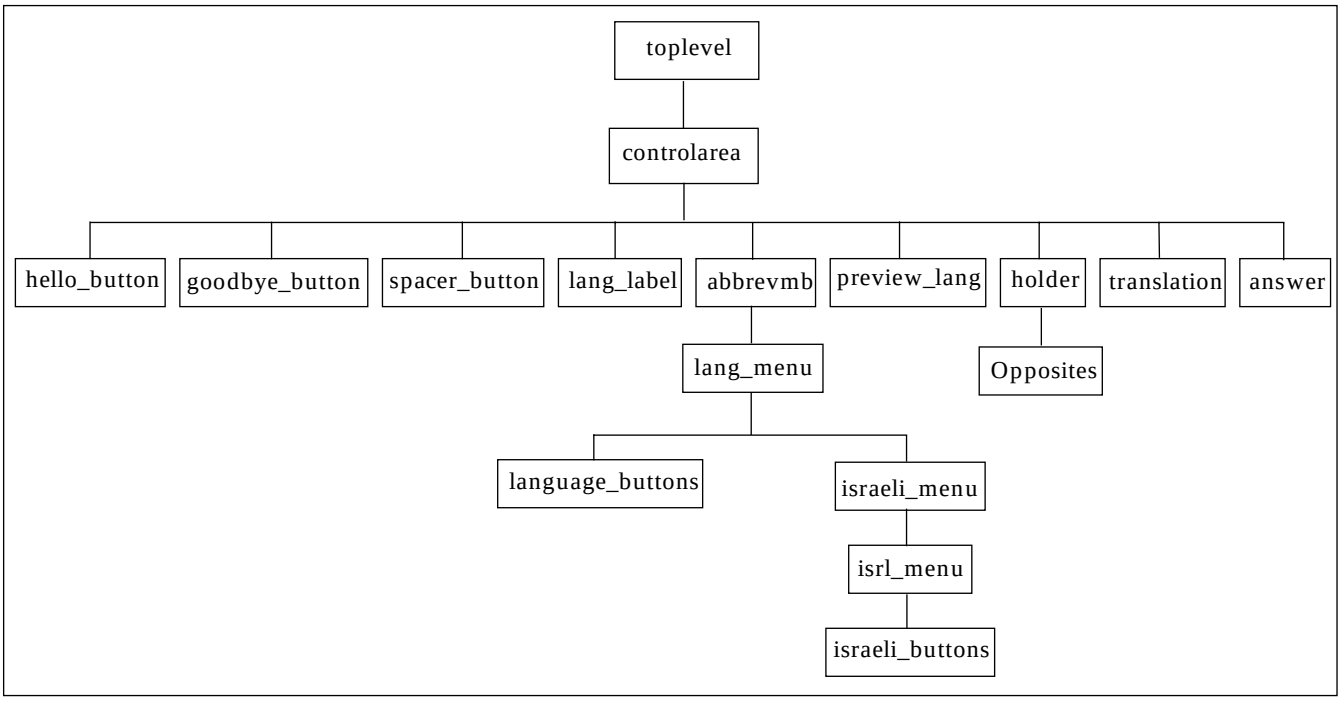


Figure 4-2 Translator.c Widget Tree Diagram

Code Example 4-1 Resource File for `Translator.c` (`Resources_translator`)

```
#######  Default Resource File  #######

hello.control_area.spacer_button.label: space
hello.control_area.spacer_button.mappedWhenManaged: FALSE
hello.control_area.lang_label.string: Language
hello.control_area.translation.string: Translation:
hello.control_area.preview_lang.string: English

*lang_label.Font: -adobe-helvetica-bold-r-normal--14-140-75-75-p-82-iso8859-1
*translation.Font: -adobe-helvetica-bold-r-normal--14-140-75-75-p-82-iso8859-1

hello.control_area.layoutType: fixedcols
hello.control_area.measure: 6

hello*holder.layoutType: fixedcols
hello*holder.measure: 1

*background:            grey85
*fontColor:             blue
*borderColor:           black
*inputFocusColor:       red
*traversalOn:           false
```

Code Example 4-2 `Translator.c`

```
/*  Translator.c program */

/* Required OLIT Headers */
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>include
#include <Xol/OpenLook.h>

/* Specific Widget Headers */
#include <Xol/ControlAre.h>
#include <Xol/OblongButt.h>
#include <Xol/MenuButton.h>
#include <Xol/StaticText.h>
#include <Xol/AbbrevMenu.h>
#include <Xol/Exclusives.h>
```

Code Example 4-2 Translator.c

```
#include <Xol/RectButton.h>

/* The following variables are global because they are used in the callback  functions, which are outside
 * of main().*/

static Widget preview_lang, hello_button, goodbye_button, answer;
char *lang_buttons[] = { “English”, “Spanish”, “Japanese”, “German”,
                       ”French”, “Swahili” };

char *israel[] = {“Yiddish”, “Hebrew”};
char *opposites[] = { “Salutations”, “Affirmations”,”Niceties”, “Gender”};

main(argc,argv)
    int     argc;
    char    *argv[];
{
    XtAppContext  app;
    Widget  toplevel, control_area,
        abbrevmb, language_buttons[XtNumber(lang_buttons)],
        israeli_buttons[XtNumber(israel)], lang_menu,
        israeli_menu, isrl_menu, lang_label, spacer_button, holder,
        Opposites[XtNumber(opposites)], translation;
    int i;
    void translation_cb();
    void new_header_cb();
    void lang_button_cb(), opposites_cb();

  /* Initialize the toolkit and create the toplevel shell.  OlSetDefaultTextFormat is for
   * internationalization and will only work with OpenWindows 3.2 or Asian OpenWindows
   * 3.1. If you are not using one of these versions, comment out this line.   */

    OlToolkitInitialize((XtPointer)NULL);
    toplevel = XtVaAppInitialize(&app, “hello”,
        (XrmOptionDescList)NULL, 0,
        &argc, argv,
        (String *)NULL,
        NULL);

/* Create application’s manager widget.  The ControlArea widget class lays out  children
 * widgets left to right in order of creation.  Thus, the first widget displays on the left, the
 * 2nd to the right of the 1st, etc.  If the layout is  specified as vertical, child widgets are
 * displayed top to bottom in order of  creation. */

    control_area = XtVaCreateManagedWidget( “control_area”,
```

Code Example 4-2 Translator.c

```
        controlAreaWidgetClass, toplevel,
        NULL);

/* Create hello button widget. */

    hello_button = XtVaCreateManagedWidget( “hello_button”,
        oblongButtonWidgetClass, control_area,
        XtNlabel, “Hello”,
        NULL);

/* Attach callback procedures to the hello_button. */

    XtAddCallback(hello_button, XtNselect, translation_cb, NULL);
    XtAddCallback(hello_button, XtNselect, new_header_cb, toplevel);

/* Create goodbye button widget */

    goodbye_button = XtVaCreateManagedWidget( “button2”,
        oblongButtonWidgetClass, control_area,
        XtNlabel, “Goodbye”,
        NULL);

/* Attach callback procedures to the goodbye_button. */

    XtAddCallback(goodbye_button, XtNselect, translation_cb, NULL);
    XtAddCallback(goodbye_button, XtNselect, new_header_cb, toplevel);

/* Create spacer button widget. This button is used to put a blank space between the goodbye
 * button and “Language:” StaticText widget. This button takes up space  but is not visible.
 * We make the button invisible by setting the OblongButton resource mappedWhenManaged to
 *  FALSE in the resources file.  */

    spacer_button = XtVaCreateManagedWidget( “spacer_button”,
        oblongButtonWidgetClass, control_area,
        NULL);

/*  Create “Language:” statictext widget.  */

    lang_label = XtVaCreateManagedWidget(“lang_label”, staticTextWidgetClass,
            control_area,
            XtNstring, “Language:”,
            NULL);

/* Create AbbrevMenuButton widget. This widget allows us to pull down a menu with
```

Code Example 4-2 Translator.c

```
 * all the language selection buttons.  */

    abbrevmb = XtVaCreateManagedWidget("abbrevmb", abbrevMenuButtonWidgetClass,
                                    control_area, NULL);

/* The AbbrevMenuButton widget is a composite widget.  That is, other widgets,
 * called subwidgets, are attached to, and part of, this widget. An AbbrevMenuButton
 * widget consists of a triangular Abbreviated Menu Button, a Current Selection
 * Widget, and a MenuShell widget. In the line below, we get a pointer to abbrevmb's
 * MenuShell widget, and assign it to lang_menu. */

    XtVaGetValues(abbrevmb, XtNmenuPane, &lang_menu, NULL);

/* Create a language preview statictext widget that will preview the language
 * selected. */

    preview_lang = XtVaCreateManagedWidget("preview_lang", staticTextWidgetClass,
                           control_area, NULL);

/* Set abbrevmb's Current Selection Widget (resource XtNpreviewWidget) to the value
 * of preview_lang. */

    XtVaSetValues(abbrevmb, XtNpreviewWidget, preview_lang, NULL);

/* Create language selection buttons (OblongButton widgets). Put language labels on
 * the buttons. XtNumber() counts the number of members in an array.  Note that if no
 * button label resource (XtNlabel) is specified in the code or a resource file, the
 * widget name is used as the label.  Thus, all our language buttons are labeled
 * with their respective widget names.  */

    for(i=0;i<XtNumber(lang_buttons);i++) {
        language_buttons[i] = XtVaCreateManagedWidget(lang_buttons[i],
        oblongButtonWidgetClass, lang_menu, NULL);

/* Attach a callback such that when the language button is pressed, the selected
 * language is displayed next to the Abbreviated Menu Button. */

     XtAddCallback(language_buttons[i], XtNselect, lang_button_cb, lang_buttons[i]);
  }

/* Create an Israeli MenuButton widget. When this button is selected then dragged
 * to the right, two more language buttons, Yiddish and Hebrew, are displayed.  */

    israeli_menu = XtVaCreateManagedWidget("Israeli", menuButtonWidgetClass,
```

Code Example 4-2 `Translator.c`

```
                                    lang_menu, NULL);

/* The MenuButton widget is another composite widget.  In addition to the menu button,
 * it also consists of a triangular mark, and a MenuShell widget. The MenuShell is
 * accessed by selecting the button, then dragging the mouse to the right or down.
 * In the line below, we get a pointer to israeli_menu's MenuShell widget, and assign
 * it to isrl_menu. */

  XtVaGetValues(israeli_menu, XtNmenuPane, &isrl_menu, NULL);

/* Create two Israeli language OblongButtons, "Hebrew" and "Yiddish" in the MenuButton
 * widget. Attach a callback such that when the Hebrew or Yiddish buttons are pressed,
 * the selected language is displayed next to the Abbreviated Menu Button.   Again, if no
 * button label resource (XtNlabel) is specified in the code or a resource file, the
 * widget name is used as the label.  Thus, all our Israeli language buttons are labeled
 * with their respective widget names. */

    for(i=0;i<XtNumber(israel);i++) {
        israeli_buttons[i] = XtVaCreateManagedWidget(israel[i],
           oblongButtonWidgetClass,
           isrl_menu, NULL);
    XtAddCallback(israeli_buttons[i], XtNselect, lang_button_cb, israel[i]);
  }

/* Create an Exclusives Widget to contain our four RectButtons (Salutations,
 * Affirmations, Niceties, and Gender).  Buttons in the Exclusives widget are
 * exclusive.  That is, only one button can be selected in an exclusives widget. */

     holder = XtVaCreateManagedWidget("holder", exclusivesWidgetClass,
                          control_area, NULL);

/* Add 4 RectButton widgets to the exclusives widget.  Attach a callback such that
 * when a button is selected, the corresponding word pair is displayed in the
 * hello and goodbye buttons. Again, the button labels take on the names of the
 * buttons, so there is no need to explicitly set the label resource. */

    for(i=0;i<4;i++) {
       Opposites[i] = XtVaCreateManagedWidget(opposites[i], rectButtonWidgetClass,
                holder, NULL);
    XtAddCallback(Opposites[i], XtNselect, opposites_cb, NULL);
  }

/* Create a statictext widget called "Translation:"  */
```

Code Example 4-2 Translator.c

```
    translation = XtVaCreateManagedWidget("translation", staticTextWidgetClass,
            control_area, NULL);

/* Create a statictext widget that displays the foreign language translation of the
 * selected button.  */

    answer = XtVaCreateManagedWidget("answer", staticTextWidgetClass,
            control_area,
            NULL);

/* realize the widgets and begin the main loop */

    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}

/*  CALLBACKS!!! */

/*  Procedure called when the hello and goodbye button are selected. Sets application header
 *  to the string in the hello/goodbye buttons.   */

void new_header_cb(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);
}

/* Procedure called when a language (lang_button) is selected. Puts the name of the
 * language next to the AbreviatedMenuButton.  */

void lang_button_cb(widget, client_data, call_data)
   Widget      widget;
   XtPointer   client_data, call_data;
{
    XtVaSetValues(preview_lang, XtNstring, client_data, NULL);
}
```

Code Example 4-2 Translator.c

```
/* Procedure called when an opposites button (Salutations, Affirmations, Niceties, and Gender)
 * is selected. When a button is selected, the corresponding word pair is displayed in the
 * hello and goodbye buttons. */

void opposites_cb(widget, client_data, call_data)
    Widget     widget;
    XtPointer  client_data, call_data;
{
    char *word;
    XtVaGetValues(widget,
       XtNlabel, &word,
       NULL);

   if (strcmp(word, “Affirmations”) == 0) {
      XtVaSetValues(hello_button,
        XtNlabel, “Yes”,
        NULL);
      XtVaSetValues(goodbye_button,
        XtNlabel, “No”,
        NULL);
    }
   else if (strcmp(word, “Salutations”) == 0) {
      XtVaSetValues(hello_button,
        XtNlabel, “Hello”,
        NULL);
      XtVaSetValues(goodbye_button,
        XtNlabel, “Goodbye”,
        NULL);
    }
   else if (strcmp(word, “Niceties”) == 0) {
      XtVaSetValues(hello_button,
        XtNlabel, “Please”,
        NULL);
      XtVaSetValues(goodbye_button,
        XtNlabel, “Thank you”,
        NULL);
    }
  else
{
      XtVaSetValues(hello_button,
        XtNlabel, “Sir”,
        NULL);
      XtVaSetValues(goodbye_button,
        XtNlabel, “Madam”,
```

Code Example 4-2 `Translator.c`

```
        NULL);
    }

}

/* Translations callback. Matches a word and language to print out the foreign language
 * translation. */

void translation_cb(widget, client_data, call_data)
     Widget        widget;
     XtPointer     client_data;
     XtPointer     call_data;
{
  char *word,*lang;
  XtVaGetValues(widget,
    XtNlabel, &word,
    NULL);
  XtVaGetValues(preview_lang,
    XtNstring, &lang,
    NULL);

 /*  English printout commands.  */

  if  ((strcmp(word, "Yes") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer,
        XtNstring, "Yes",
        NULL);
  if  ((strcmp(word, "No") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "No", NULL);
  if  ((strcmp(word, "Hello") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "Hello!", NULL);
  if  ((strcmp(word, "Goodbye") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "Goodbye!", NULL);
  if  ((strcmp(word, "Thank you") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "Thank you", NULL);
  if  ((strcmp(word, "Please") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "Please", NULL);
  if  ((strcmp(word, "Sir") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "Sir", NULL);
  if  ((strcmp(word, "Madam") == 0) && (strcmp(lang, "English")) == 0)
      XtVaSetValues(answer, XtNstring, "Madam", NULL);

/*  Translation Tables.  When word and language button match is found,
 *  the English translation is printed on screen */
```

Code Example 4-2 Translator.c

```
/*  Spanish translations.  */

  if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Sí”, NULL);
  if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “No”, NULL);
  if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Hola!”, NULL);
  if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Adiós!”, NULL);
  if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Gracias”, NULL);
  if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Por Favor!”, NULL);
  if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Señor”, NULL);
  if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “Spanish”)) == 0)
      XtVaSetValues(answer, XtNstring, “Señora”, NULL);

/*  Japanese translations.  */

  if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “Hai”, NULL);
  if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “iie”, NULL);
  if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “Moshi-moshi!”, NULL);
  if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “Sayonara”, NULL);
  if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “Domo arrigato”, NULL);
  if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “Dozo”, NULL);
  if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “-san appended to name, e.g., \
      Honorable Clinton-san”, NULL);
  if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “Japanese”)) == 0)
      XtVaSetValues(answer, XtNstring, “-san appended to name, e.g., \
      Honorable Hillery-san”, NULL);

/*  German translations.  */

  if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “German”)) == 0)
```

Code Example 4-2 `Translator.c`

```
        XtVaSetValues(answer, XtNstring, “Ja”, NULL);
    if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “nein”, NULL);
    if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “Hallo!”, NULL);
    if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “Auf wiedersehen!”, NULL);
    if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “Dankeschön”, NULL);
    if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “Bitte”, NULL);
    if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “Herr”, NULL);
    if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “German”)) == 0)
        XtVaSetValues(answer, XtNstring, “Frau”, NULL);

/*  French translations.  */

    if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Oui!”, NULL);
    if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Non!”, NULL);
    if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Allo!”, NULL);
    if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Adieu!”, NULL);
    if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Merci!”, NULL);
    if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “S’il vous plâit”, NULL);
    if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Monsieur”, NULL);
    if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “French”)) == 0)
        XtVaSetValues(answer, XtNstring, “Madame”, NULL);

/*  Swahili translations.  */

    if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Naam”, NULL);
    if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Siyo”, NULL);
    if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Jambo!”, NULL);
    if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “Swahili”)) == 0)
```

Code Example 4-2 Translator.c

```
        XtVaSetValues(answer, XtNstring, “Kwaheri!”, NULL);
    if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Asante”, NULL);
    if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Pendeza”, NULL);
    if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Bwana”, NULL);
    if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “Swahili”)) == 0)
        XtVaSetValues(answer, XtNstring, “Bibi mkubwa”, NULL);

/*  Yiddish translations.  */

    if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “Ya”, NULL);
    if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “Neyn”, NULL);
    if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “Sholom a ley-khem!”, NULL);
    if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “A gut-n tog!”, NULL);
    if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “Zayt a-zoy gut”, NULL);
    if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “A dank”, NULL);
    if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “Givar”, NULL);
    if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “Yiddish”)) == 0)
        XtVaSetValues(answer, XtNstring, “Givaret”, NULL);

/*  Hebrew translations.  */

    if  ((strcmp(word, “Yes”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Ken”, NULL);
    if  ((strcmp(word, “No”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Lo”, NULL);
    if  ((strcmp(word, “Hello”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Shah-lom!”, NULL);
    if  ((strcmp(word, “Goodbye”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Lehitra-ot!”, NULL);
    if  ((strcmp(word, “Thank you”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Bevakasha”, NULL);
    if  ((strcmp(word, “Please”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Toda raba”, NULL);
    if  ((strcmp(word, “Sir”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
```

Code Example 4-2 Translator.c

```
        XtVaSetValues(answer, XtNstring, “Adon”, NULL);
    if  ((strcmp(word, “Madam”) == 0) && (strcmp(lang, “Hebrew”)) == 0)
        XtVaSetValues(answer, XtNstring, “Geveret”, NULL);
}
```

Glossary

application context

A pointer to an opaque data structure containing all the information the toolkit maintains for one application.

attributes

See *resources*.

callback procedures, callbacks

Procedures executed when some user action occurs on a widget.The procedures are typically written as part of the applications, but invoked by the toolkit.

callback list, callback resource

List of callback procedures that are executed when some user action occurs on a widget. The callback list is referenced as an OLIT-defined callback resource.

child widget

1. In the OLIT class hierarchy (see Figure 1-3), a *subclass* of a widget is referred to as the child widget.

2. In an application, a child widget is a widget that is owned and managed by a parent widget. Parent widgets manage the size and location of their children, and control input to their children by controlling the input focus. An application organizes widgets into a hierarchy of children and parents, the root of which is the top level shell widget created by `XtVaAppInitialize()`.

class

See *widget class.*

client-server computing model

A computing model in which a client (application) receives user input and sends output to a display server (a program which controls a workstation's display) rather than directly to a workstation's hardware.

composite widget

Widgets composed of other *subwidgets*. For example, the MenuButton widget consists of a MenuButton plus MenuShell.

display

1. Hardware: a computer's monitor or CRT.
2. Software: a single display server process, which is capable of displaying to multiple CRTs.

Inheritance

When a widget subclasses another widget, the subclassed widget *inherits* operating characteristics and the resource set of its superclass.

instantiation

The process of creating a particular widget from a particular widget class. The term *widget instance* refers to a specific widget as opposed to a widget class.

manager widgets

A class of widgets that contain and manage other widgets. ControlArea, Exclusives, Form, and RubberTile are typical manager widgets. Some manager widgets, such as CheckBox, Caption, FileChooser, and ScrolledWindow, are both viewable and perform user input/output functions.

name

A string associated with a widget instance. Used to specify resource values in a resource file.

object

A programming unit consisting of unchangeable code and changeable data that create a programmatic entity with which to construct programs. Widgets are objects.

OLIT

An X Window System-based® widget set and library used to create applications using the *OPEN LOOK*® graphical user interface.

parent widget

1. In the OLIT class hierarchy (see Figure 1-3), a widget's *superclass* is referred to as the parent widget.
2. In an application, all widgets are *children* of some parent widget. Parent widgets manage the size and location of their children and control input to their children by controlling the input focus. An application organizes widgets into a hierarchy of children and parents, the root of which is the top level shell widget created by `XtVaAppInitialize()`.

primitive widgets

A class of displayable widgets that usually perform some type of information display or user interaction function. Buttons, Gauges, Scrollbars, Sliders, and DropTargets are all primitive widgets.

resources

Named, settable, attributes of a widget such as button label, background color, font type, or widget position.

server

1. Software: a program which controls a workstation's display.
2. A computer that servers other node computers by storing/retrieving files, sending/receiving e-mail, etc.

subclass

A widget class created from another widget class. A subclassed widget is created by modifying and specializing another widget class called the *superclass*. The subclass inherits some or all of the characteristics of it's superclass. Class architecture and inheritance make it easier to create new widgets because as subclasses use much of the same code and declarations as its superclass.

subwidget

A widget that is a component of another widget. The MenuButton widget contains one subwidget, a MenuShell, that is accessible through the `XtNmenupane` resource.

superclass

A widget class which is modified and specialized to create another widget class (a *subclass*). The subclass inherits some or all of the characteristics of the superclass. Class architecture and inheritance make it easier to create new widgets because subclasses use much of the same code and declarations as its superclass.

widget

User interface elements like buttons, scrollbars, control areas, text edit areas, etc. Programmatically, widgets are specialized X windows implemented as data structures. When widgets are created by some X Toolkit Intrinsics function, they are returned as an opaque data handle and assigned to a variable called a widget identifier.

widget class

Refers to the widget's type. Class defines the characteristics and set of operations that can be performed on a class of widgets. ControlArea, OblongButton, and Shell are widget classes. See Figure 1-3.

widget class hierarchy

The hierarchy of widget superclasses and subclasses. See Figure 1-3.

widget instance

Refers to a specific widget instance as opposed to a widget class. See instantiation.

Widget set

A family of widgets used together to produce a unified user interface. Athena is one such widget set. OLIT is another.

Xlib

The C language interface to the X protocol.

X Window System Protocol

The computer protocol by which clients communicate with the X server and vice versa. Also referred to as the X protocol.

X Server

A program which controls a workstation's display. The X Server handles output from an application (also called the *client*) to the screen(s), and sends input from the keyboard or mouse to the appropriate client for processing.

X Window System Toolkit

An X consortium standard that provides the structure and library functions for creating and assembling widgets into a user interface. Often referred to as the X Toolkit, Xt Intrinsics, Intrinsics, and Xt.

Index

X