OLIT Program Structure 2

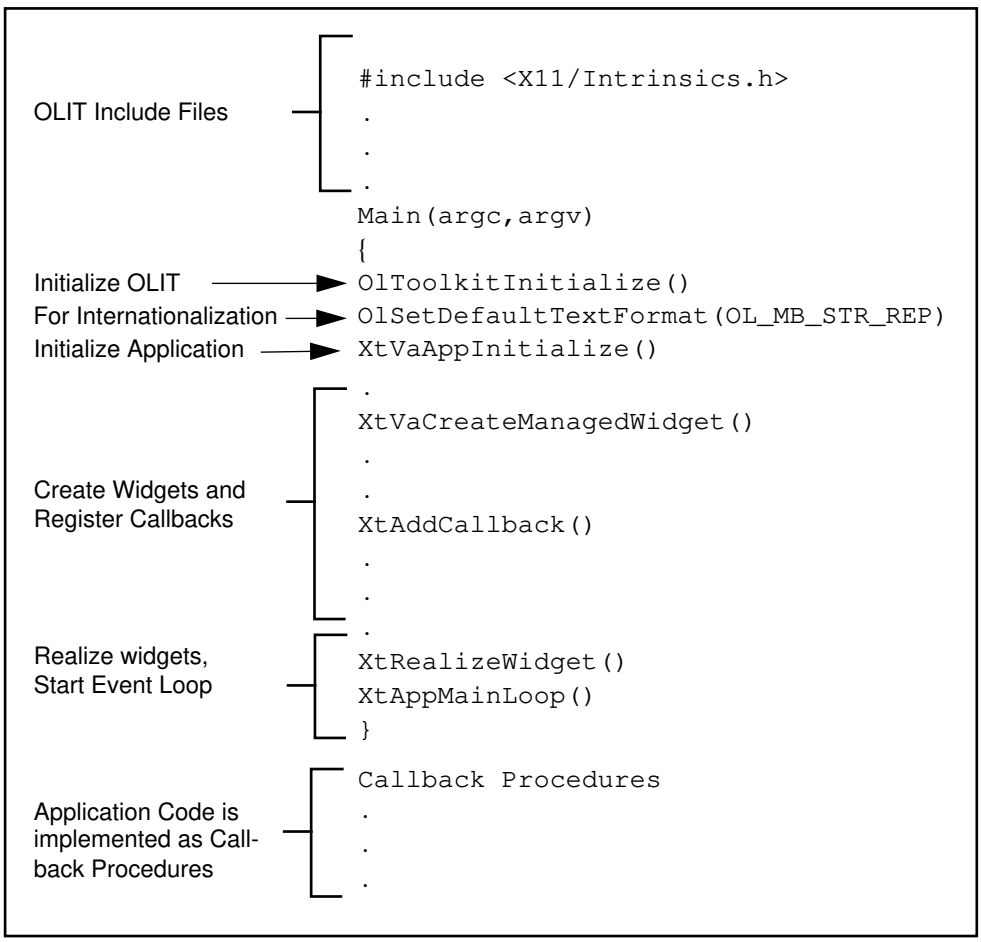
Overview

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.

2

`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*.

2

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.

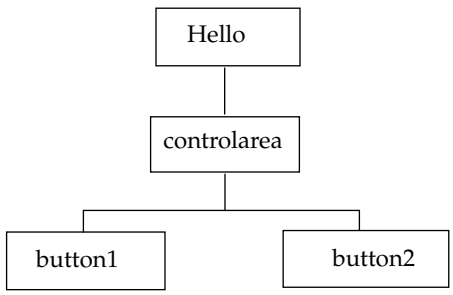


Figure 2-2 `hello.c` Widget Instance Tree

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.

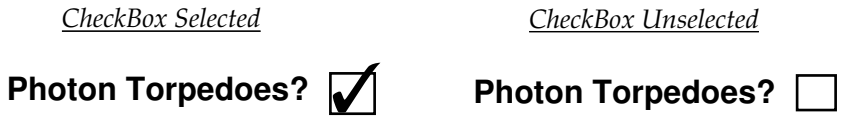


Figure 2-3 CheckBox Widget

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