BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-93-M13

"XEMS: An X-based Event Messaging System"

by

Eddy Wong

XEMS: An X-based Event Messaging System

Eddy Wong

Copyright © 1992, 1993 Eddy Wong
Copyright ©1993 Brown University

XEMS: An X-based Event Messaging System

Eddy Wong

Department of Computer Science
Brown University
ew@cs.brown.edu

Cambridge Technology Partners, Inc.
ewong@ctp.com

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science in the
Department of Computer Science at Brown University

April 29, 1993

This research project by Eddy Wong is accepted in its present form by the Department of Computer Science at Brown University in partial fulfillment of the requirements for the Degree of Master of Science.

Professor Steven P. Reiss
Advisor

4/29/93
Date

Abstract

In this paper, we describe XEMS, an X-based Event Messaging System. This system provides the framework for interactive duplication, storage and manipulation of X Window user-generated input events. XEMS is a client-server system that relies on X interclient communication to send events from an application to a storage daemon and vice versa. Since XEMS is build above X, it is portable across platforms and widget toolkits. For this project, we have implemented a regression tester using XEMS. XEMS can also be used for logging, demonstration, and multiplexing.

Contents

1. Introduction **1**

2. The X Window Event Driven Model **1**

3. System Overview **2**
3.1. Issues 2
3.2. The Recording Process 3
3.3. The Playback Process 4
3.4. XEMS Architecture 6

4. The Client Module (CM) **8**
4.1. Architecture 8
4.2. Data Flow 11

5. The Event Daemon (ED) **14**
5.1. User Interface 14
5.2. Architecture 14
5.3. Data Flow 20

6. The Event Messaging Protocol **21**
6.1. Target Application Selection Mechanism 22
6.2. Window Identifier Number Encoding and Decoding 22
6.3. ClientMessage Event Structure Format 24
6.3.1 Changing CM States 25
6.3.2 Storing Time and Name Information 26
6.4. Event Filtering Scheme 27

7. User's Guide **28**
7.1. Installation 28
7.2. Selecting a Target Application 28
7.3. Changing State 28
7.4. Queue Management 29
7.5. Turning Messages On and Off 29
7.6. Filtering Events 29
7.7. Saving and Retrieving Events 29
7.8. Resetting and Quitting 29
7.9. Scenario 29

8. Conclusions and Outlook **30**

9. References **31**

10. Appendix **32**

Table of Figures

1. The Recording Process 4

2. The Playback Process 5

3. The XEMS Architecture 7

4. Overview of the Client Module Data Flow 9

5. Components of `XemsModifyReceiver()` 10

6. Components of `XemsModifySender()` 11

7. Data Flow of `XemsModifyReceiver()` 12

8. Data Flow of `XemsModifySender()` 13

9. Event Daemon User Interface 14

10. Components of EdModifyReceiver() 15

11. Components of the Event Daemon Sender 16

12. Components of the Event Daemon User Interface 17

13. Components of the Filter 19

14. Components of the Converter 19

15. Data Flow Diagram of `EdModifyReceiver()` 20

16. An example of Window Identifier Number Encoding and Decoding 23

17. Format of a `ClientMessage` event containing a change of state message 25

18. Format of Time and Name trailer messages 26

19. Example of a Filter Setup operation 27

1 Introduction

The X-based Event Messaging System (XEMS) provides a mechanism to intercept user generated events and to ship them between X clients using the X Window interclient communication routines. XEMS has its own protocol to send event structures and related information across X clients. In addition, XEMS provides an interactive framework to capture, replay and filter events through a user interface. Using XEMS we have implemented a regression tester, a system that captures mouse and keyboard events from an X application (the "Target Application"). These events can then be stored and played back later, even on another instance of the same application.

In the past, there have been efforts to capture user generated events with specialized X servers[1] that trap events at a lower level. This approach is consequently device dependent and obviously not very portable. In comparison, XEMS relies its functionality on interclient communication provided by X, specifically by the `XSendEvent`[2] routine. By preserving the X server integrity and by staying above the Xlib layer[3], we provide a robust and portable system that relies on X Window portability and does not deal with the specifics of device drivers. In addition, XEMS provides a uniform protocol for applications that use Athena, Motif, OpenLook or any other Xt-based toolkit.

2 The X Window Event Model

The X Window system is a mouse-driven windowing system which responds to *events*. An event is produced by a keypress, by a mouse click or motion, or by interaction between X programs. Events occur at any time, and need to be stored as soon as they happen. The *Event Queue* is the data structure dedicated to store the information about events. X provides functions to take events from this queue. Once an event is taken from the queue, the application program must generate the appropriate response depending on the type of this event.

In order process events, every X program must have an *Event Loop*.This loop is implemented as an infinite `while` or `for` loop, which calls two routines: one that takes events from the event queue; and another that branches out according to the type of event and generates the appropriate response, in the case of widget toolkits, by dispatching events to the widget that originated the event. The purpose of the infinite loop is not that of polling, on the contrary, the

1. XTrap [2] and XTest [11] are examples of this approach. The source code and documentation for these systems are available via ftp from export.lcs.mit.edu in the /contrib and /pub directories respectively.
2. Section 8.4. of [7] provides further details on the `XSendEvent` library routine.
3. Figure 1-3 in page 9 of [7].

program is put to sleep while no events occur and waken up as soon as the event queue is filled. The Xt library provides the routine that implements this event loop and its coded as follows:

```
void XtAppMainLoop(app)
XtAppContext app;

{
    XEvent event;
    for (;;) {
        XtAppNextEvent(app, &event);/*take event from queue*/
        XtDispatchEvent(&event);    /*dispatch to widget*/
     }
}
```

`XtAppNextEvent` is the routine that removes events from the event queue and `XtDispatchEvent` is the one that branches out and dispatches events to the appropriate widget. Notice that the event information is passed between the two routines through the `event` variable.

Event information is stored in an *Event Structure*[1], which is a record containing items such as the type of the event, the window and display that originated the event, the system time and other *ad-hoc* information particular to the type of the event. An instance of this event structure is depicted in the `XtAppMainLoop` routine as the `event` variable and it has the type `XEvent`.

The goal of XEMS is to capture these event structures between the time they are taken off the event queue and the time of dispatching. Once this is accomplished, the event information can be sent to another X client via the X network.After this application program can continue normally. However, storing only event structures is insufficient in order to reproduce them later. Additional information must be associated to each event, as we will see in the next section.

3 System Overview

3.1 Issues

X Window events can be categorized into two types: user generated and window-manager generated. User generated events are the ones created directly by user input. For example, a mouse click, which creates an event structure containing information on the position of the cursor, time, display and window identifier number and so forth. The application program must then respond to this event, for instance by popping up a window or by drawing a dot. On the other hand, events generated by the window manager are created when the user interacts with the user interface of the window manager. They are informational in nature, and do not require a response

1. Chapters 2 and 8 of [7] provide more detailed information about events.

from the application program, since these events are created after the action of the window manager on the application. For example, moving and resizing are events generated by the user interface of the window manager after the action of moving or resizing has been executed. Then the window manager sends a `ConfigureNotify` event to the application; however, the application does not take any further action in response to this event.

The backbone of our interclient communication between the target application program and the storage daemon (the process that will store and manage the stored events) is the `XSendEvent` X library routine. One of the arguments that `XSendEvent` takes is the identifier number of the window where the event is to be sent. This is of great advantage in order to reproduce events later, since the identifier number of the window that generated the event is stored in the event structure, and since each widget has a window identifier number associated with it. However, window identifier numbers are assigned at run time and might be different from instance to instance of an application. Thus, this poses a problem if we were to store the events to a file, because we will be unable to find the window identifier number associated to the widgets of a new target application that is different from the one that created the stored events. We will discuss a solution for this problem in the following sections.

3.2 The Recording Process

Events are captured from the target application between the time they are taken off the event queue and the time that they are dispatched to the widgets. The event structures are sent using the `XSendEvent` routine to a storage daemon application that we will call *Event Daemon* (ED). The ED will recognize the events coming from the target application and will proceed to store them in a queue. In addition to the event structure, two `ClientMessage` event structures are sent synchronously with each event. One containing the system time when the event was captured and the other the name of the widget that originated the event. The ED provides a graphical user interface that allows the user regulate the flow of events and control the states in the target application[1].

`ConfigureNotify` events have to be handled as special cases. As we mentioned before, these event structures are issued by the window manager after a move or resize action. They provide information and usually do not require any response from the application. However, `ConfigureNotify` events generated by a resize action only include information about the width and height of the window that has been resized, but not the absolute coordinates of the top left corner of the window. Thus, this information has to be requested through the X routines `XGetGeometry`, `XGetWindowAttributes`, and `XTranslateCoordinates` and included in the event structure, so that window geometry information is complete and

1. The target application must be modal in order to handle recording and playback.

reproducible.

Figure 1 depicts the recording process between two X clients: the target application and the Event Daemon. The events are sent from the application to the ED using the interclient communication routines provided by X, which we have mentioned previously. To be able to intercept the events on the application side, the application has to either use an event loop routine provided by XEMS (`XemsMainLoop`) or link to a modified Xt library also provided by XEMS. In both versions, the application is connected to a module that we will call the *Client Module* (CM). This module is in charge of sending the events to the ED and detect events coming from it.

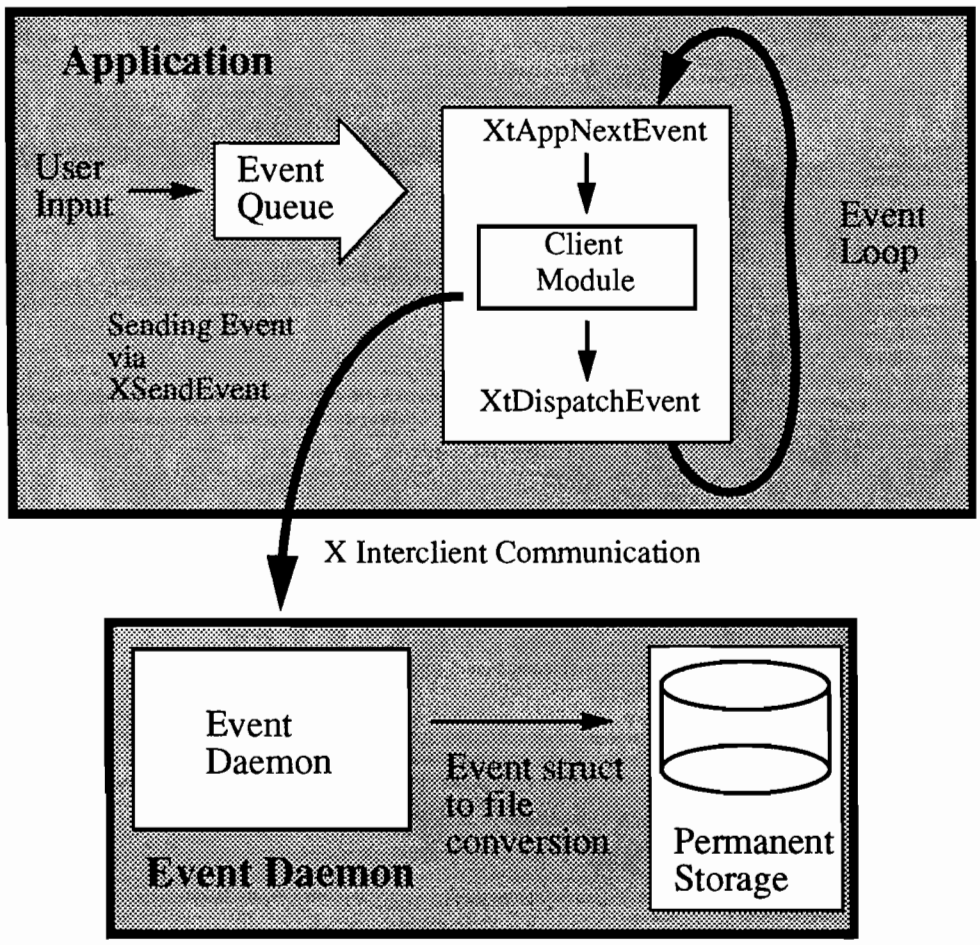


Figure 1: The Recording Process

3.3 The Playback Process

The Playback Process is a more complicated one. Once the events are stored in the ED

side, they are arranged in a queue (the "EdQueue") in main memory, so that they are ready to be sent to the application, thus simulating user input. Additional information such as the name of the widget that originated the event and the system time when the event occurred is associated with every event. This information is obtained from the two `ClientMessage` structures that were sent synchronously after each event during the Recording Process.

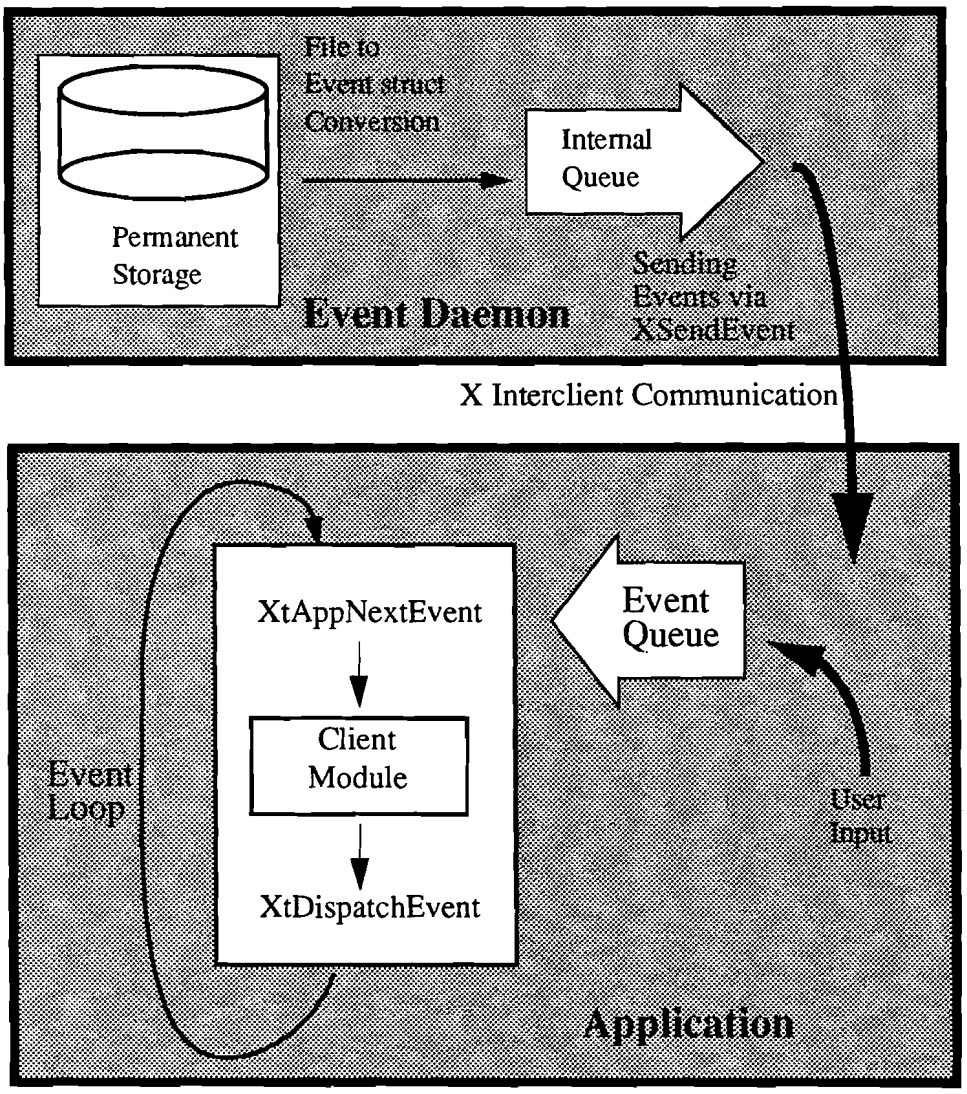


Figure 2: The Playback Process

To start reproducing user input, we take events one by one from the EdQueue, following the timing elapsed from one event to another[1]. Each event has its `window` (the window identifier number) field encoded to uniquely identify the event as coming from the Event Daemon. This

1. The timing simulation is implemented with the use of timeouts. For example, if there are 2 seconds elapsed from one event to another, we set up a timeout of 2 seconds after the first event is sent. Once the 2 seconds have elapsed, a routine will be called to send the second event.

event is then sent through the X network using the `XSendEvent` routine, from the ED to the toplevel widget of the target application. Once on the application side, the CM would detect this event as coming from the ED by looking at the encoded `window` field. This field will contain an encoded identifier that X would never generate. Then the CM will proceed to decode the modified window identifier. After the decoding process, the CM recreates the window identifier number as if it were created by user input[1].

Another way to decide where an event should be dispatched to on the application side is to send a `ClientMessage` structure from the ED to the CM containing the name of the widget to which we want to send the event. This name will be looked up in a hash table[2] that will map widget names to window identifier numbers. After the mapping is done, the window identifier number will be stored in the `window` field of the event structure, so that it will be dispatched by `XtDispatchEvent` to the widget that supposedly originated the event.

3.4 XEMS Architecture

XEMS, as described before, consists of two main X Window client applications: The Target Application, the process or application whose events we would like to capture and play back; and the Event Daemon (ED) application, which is responsible for storing and managing the events that we capture. These two applications are depicted in *Figure 3*.

The target application has the Client Module (CM) attached to it. This is accomplished by either using the main loop routine provided by XEMS (`XemsMainLoop`) or linking the application to a modified X Toolkit Intrinsic library (libXt.a). The Xt library can be also modified and made available as a shared library, in this case, potential target applications are not required to be re-linked. By using the shared library approach, we make every X client that uses the Xt main loop a potential target application. `XemsMainloop` would be more suitable in situations where the source code for the target application is available, whereas the shared library approach works for applications that might be only available in executable format.

The CM is in charge of deciding whether an event should be sent to the ED. For that reason, the CM must be modal, and therefore has three states: Normal, Record, and Play. These states are set by the user through the ED user interface. The ED communicates this information to the CM via an X Window interclient mechanism. In Record mode, the CM sends all user and window-manager generated events to the ED. In Play mode, the CM is ready to receive events

1. We are assuming that the order of widget creation is consistent across instances of the applications that created the stored events and the application that will receive them. If so, X will assign window identifier number having the same low order bits to the widgets in the application.
2. This hash table would be created when widgets are realized, this; however, requires modification of the Xt library.

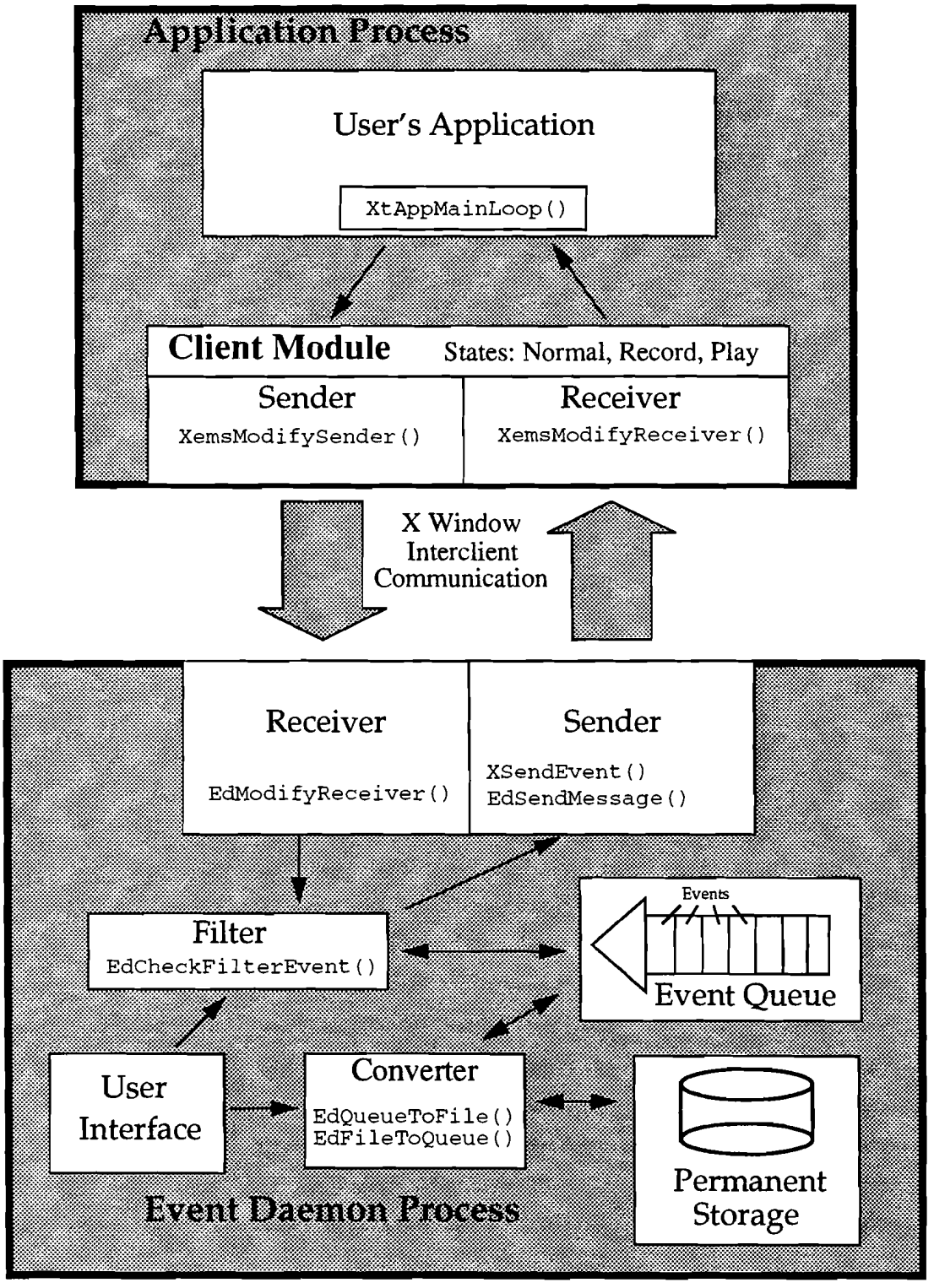


Figure 3: The XEMS Architecture

from the ED. And finally, in Normal, mode, the CM ignores all events coming from the ED, except the ones that change states (`ClientMessage` events). The CM consists of two modules to handle the traffic of events: The Sender, which is the module that sends the events and the Receiver, the one that receives events.

Analogously, the ED has also a Sender and a Receiver. However; the sender does not reside inside the event loop, since actions are generated through the ED user interface. The Receiver identifies events coming from the CM by decoding the window identifier number stored in every `XEvent` structure. In addition to events that were created by the user on the target application user interface, the ED receives also two `ClientMessage` events that store additional information about the event that preceded them. One stores the system time when the event was captured, and the other the name of the widget that originated the user-generated event on the target application.

Once the events are received on the ED side, they are stored in memory, specifically in a queue analogous to an event queue, called EdQueue. The EdQueue can then be traversed to reproduce the events or stored permanently in a file for future retrieval. It can be also manipulated through the ED user interface. The queue can be cleared, scanned, stepped and reset.

The ED, has two storage units, one is the EdQueue that resides main memory and the other is the Permanent Storage residing in the Unix the file system. Events can be transferred from Permanent Storage to the EdQueue and vice versa, via the Converter, which consists of a group of routines that convert `XEvent` structures to a file and vice versa. All events coming and leaving to and from the ED are passed through the Filter, which eliminates events that the user wants to filter out. This functionality is provided by the ED User Interface, which also allows the user to change the CM state, manipulate the queue and manage message printing. The interaction among these units is depicted in *Figure 3*.

4 The Client Module (CM)

4.1 Architecture

The Client Module (CM) is connected to the target application via the `XemsModifyEvent` routine which is embedded in the main loop. `XemsModifyEvent` calls three routines: `XemsModifySetup`, which stores initial connection information like display and window identifier numbers; `XemsModifyReceiver`, which is in charge of receiving events from the ED and `XemsModifySender`, which sends the events to the ED. The data flow among these units is depicted in *Figure 4*. To store global information, which is used across the various units of the CM, we use a structure called `CmInfo`. This structure contains display, window, state and coordinate information of both the CM and the ED. This information is

necessary in order to establish a connection between the two X client processes.

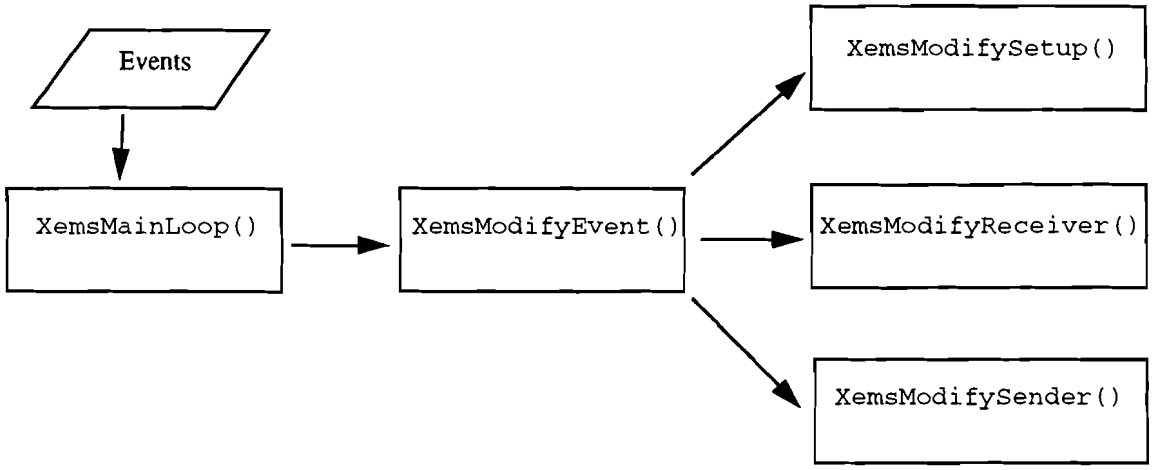


Figure 4: Overview of the Client Module Data Flow

The Receiver, as depicted in *Figure 5*, is represented by the `XemsModifyReceiver` routine. It is in charge of receiving the events from the ED. To be able to distinguish events that are coming from the ED, the CM uses the routine `XemsModifyId` which decodes the window identifier number stored in every stored event structure. This number has been previously encoded by the ED. After the number is recognized as coming from the ED and decoded, there can be three types of events that the CM Receiver can process: `ClientMessage` events, window-manager generated events and other events that do not fall under these categories.

`ClientMessage` events coming into the CM involve a change of state[1]. This is handled by the `XemModifyReceiverClient` routine. Window-manager generated events currently supported are `ConfigureNotify`, which is issued for move and resize events; `MapNotify`, which is issued when a window is mapped or displayed on the screen and `UnmapNotify`, which is the opposite of `MapNotify`, when the window is popped down. These window-manager generated events by themselves do no elicit any actions if they are dispatched to a widget. For that reason, we must extract information from the event structure and call the appropriate X routine that will correspond to the action that made the window manager create these events. The routines that handle these events are: `XemsModifyReceiverConfigureNotify`, `XemsModifyReceiverMapNotify`, and `XemsModifyReceiverUnmapNotify`. These routines int turn, call the X routines `XMoveResizeWindow`, `XMapWindow` and `XUnmapWindow` respectively.

The Sender, as depicted in *Figure 6*, is contained in the routine `XemsModifySender`. It

1. To find out more about change of state see section *6.3.1, Changing CM States.*

is in charge of packaging the events to be sent to ED with the information that allows the events to be replayed. The decision on whether to send events is based on the current CM state, which is checked before any event is sent to the ED.

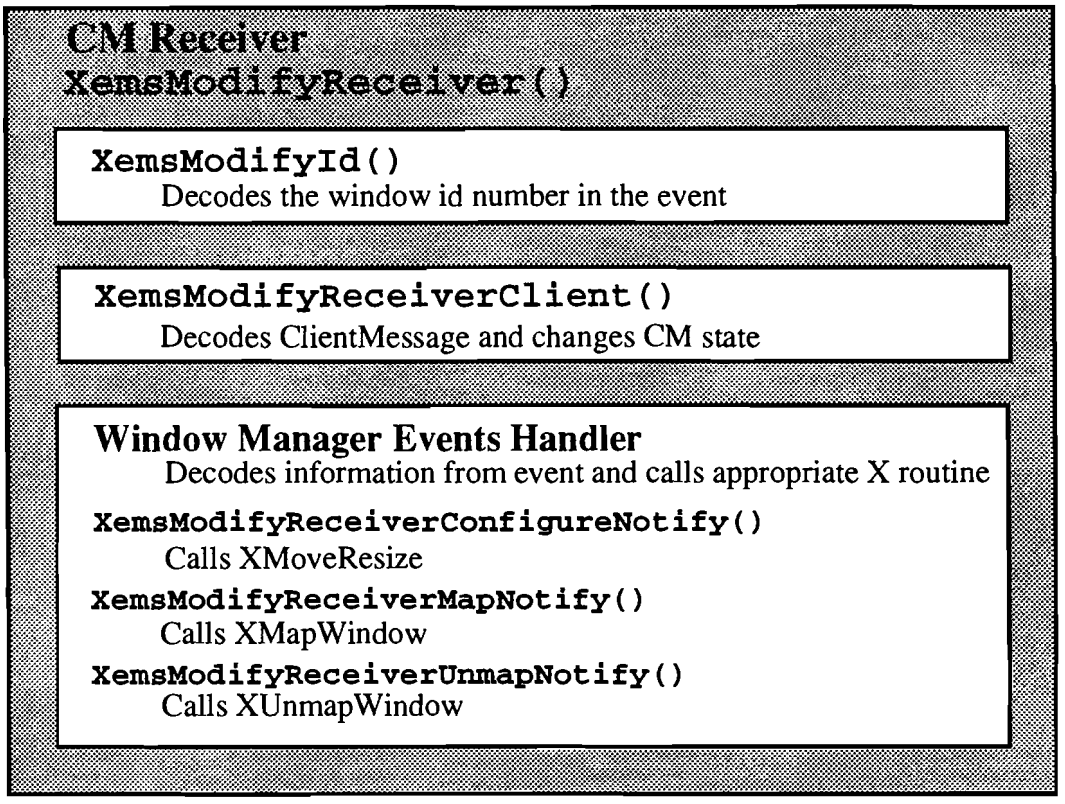


Figure 5: Components of `XemsModifyReceiver()`

The Sender must pre-package the events that are being sent out. `ConfigureNotify` events must be processed in a special manner. A `ConfigureNotify` event generated by a resize action does not include the top left coordinates of the window that originated the event; therefore, we must retrieve these coordinates through the `XGetGeometry`, `XGetWindowAttributes` and `XTranslateCoordinates` X routines. This has to be done with every `ConfigureNotify` event because X does not provide any explicit information on whether the event was created by a move or resize action. Resize events have the top left corner coordinates[1] always set to zero.

The other main part of the Sender is the actual sending mechanism, embodied by the `XemsSendEvent` routine. `XemsSendEvent` sends three event structures for every event that

1. The top left corner coordinates are stored in the fields x and y of the event structure.

is user or window-manager generated. One is the event itself, and the others are trailer `ClientMessage` events containing information on the system time when the event was captured and the name of the widget that generated the event. All these three events are sent using the X routine `XSendEvent` provided the CM state is set to Record mode.

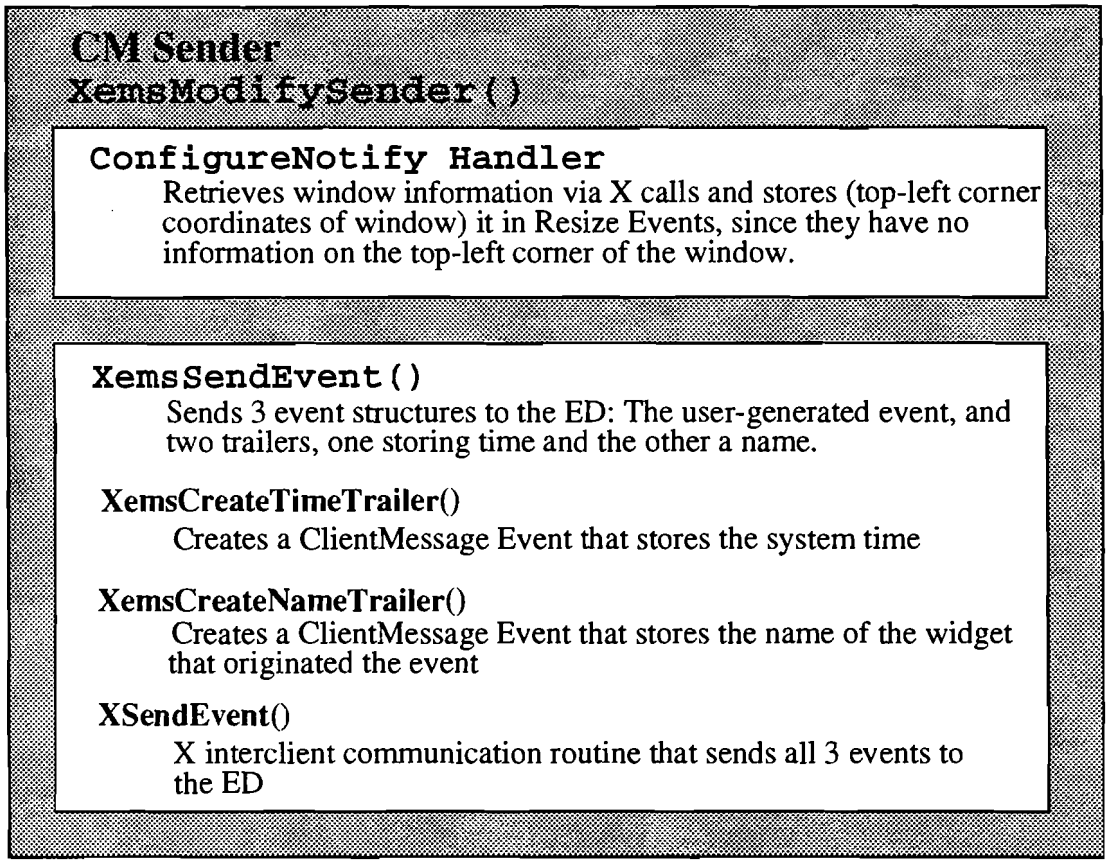


Figure 6: Components of `XemsModifySender()`

4.2 Data Flow

The Client Module actions are driven by the events that go through the main loop. These events can be user or window-manager generated from the target application or coming from the ED. The CM must filter these events and decide on the appropriate actions to take depending on the CM state and the origin of the event. To handle this process, as we explained in the previous section, CM consists of two major parts: The Receiver and the Sender. In the following paragraphs, we will describe step by step the process that an event goes through within each of these modules.

The Receiver looks first at the window identifier number stored in every event structure,

then it decides whether the event comes from the ED. If it does, the event might have been a user or window-manager generated event which will be left for the Sender to process. On the other hand, if it comes from the ED, it will decode the window identifier number so that the event looks as if it was user or window-manager generated. Following, it will check on the type of event. If it is a ClientMessage event, it involves a change of the CM state and it will make the call to execute the change. If the event is of another type, it will check whether the CM state was set to Play mode. If it is in Play mode, it would check if the event is user generated or window manager generated. If it is user generated it just lets it go, since the widget will handle the event by itself. Otherwise, if it is a window-manager generated event, it will call the appropriate routine to simulate the action that generated such event. If the CM state is set to Normal or Record mode, it will ignore events coming from the ED, except ClientMessage events. Here is a data flow diagram of the process that we just described:

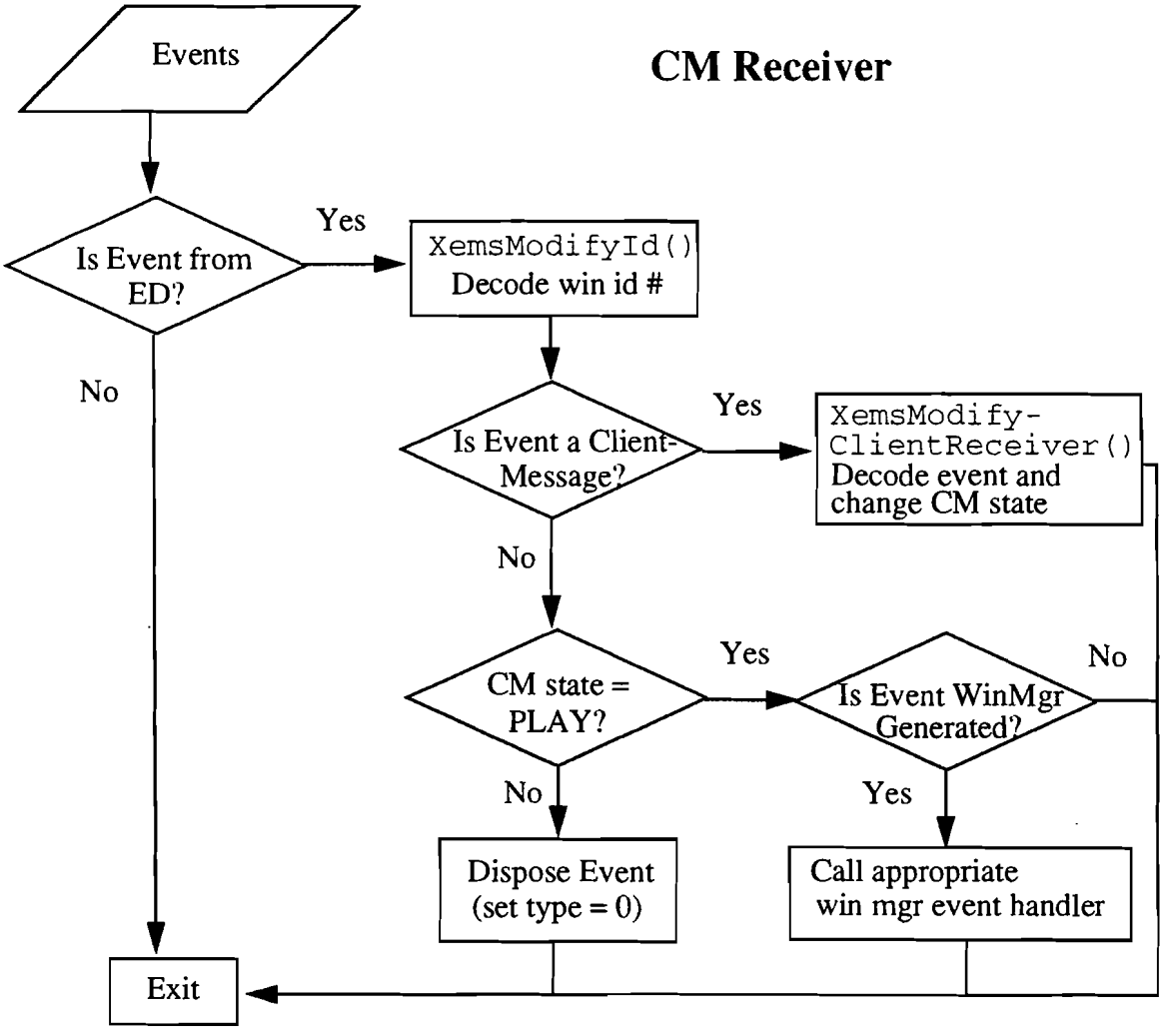


Figure 7: Data Flow of XemsModifyReceiver()

The Sender handles events originated within the target application and sends the events to the ED, if the CM state is set to Record mode. The Sender must do a special alteration to `ConfigureNotify` events, as we have discussed before. Once the event is identified as a `ConfigureNotify` event, we must know whether it was created by the target application[1]. If it was not, then it was sent by the ED and has been already processed by the Receiver. Otherwise, it proceeds to retrieve the top left corner coordinate information and store it in the event structure. Finally, the Sender checks the CM state. The state must be set to Record mode, so that the Sender can send the event to the ED, otherwise the event just trickles down to the widget dispatcher. This is the data flow diagram of the Sender process:

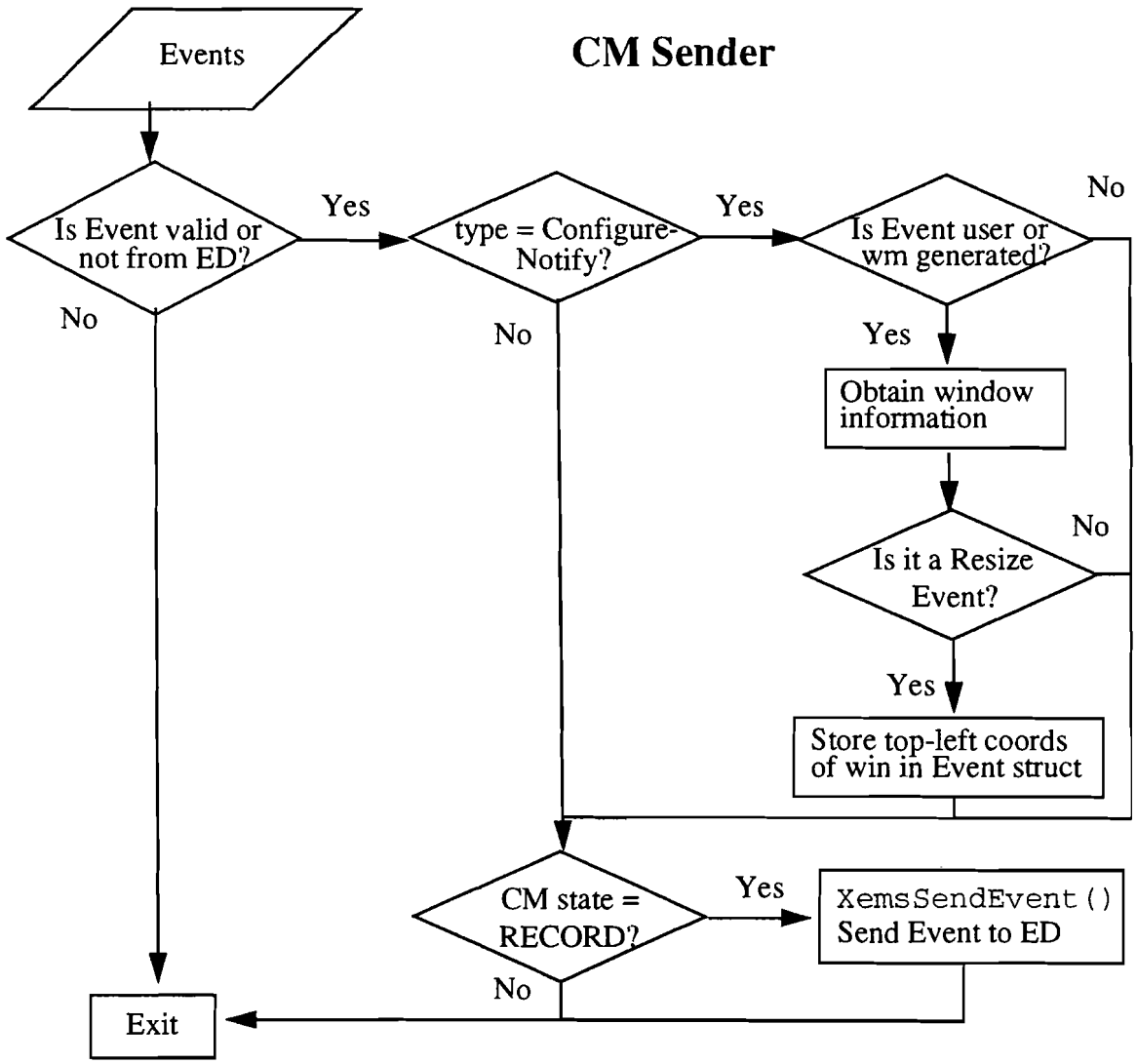


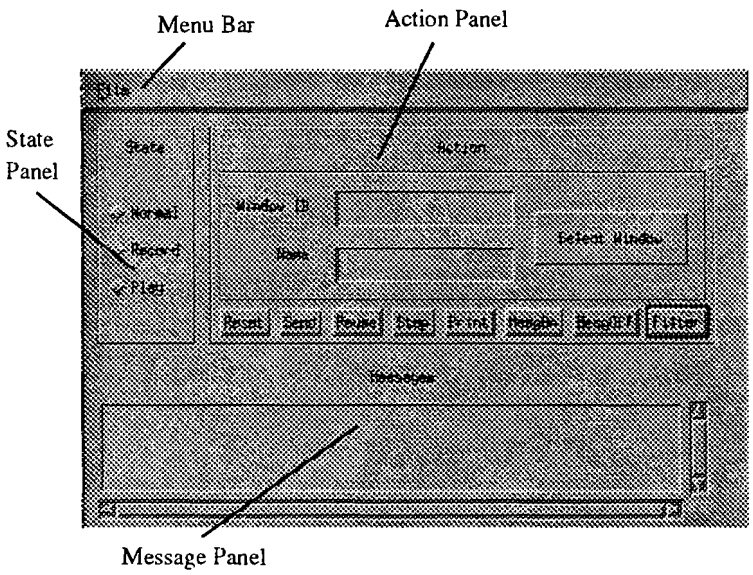
Figure 8: Data Flow of `XemsModifySender()`

1. This implies being user or window-manager generated.

5 The Event Daemon (ED)

5.1 User Interface

Although "daemon" usually implies a background process, the "Event Daemon" is process that hides the peculiarities of the X interclient communication with the Client Module through a graphical user interface. This GUI provides mechanisms to save and retrieve events, filter events, change the state of the CM, manipulate the EdQueue, manage message printing and inform the user about the status of the system. The user interface has two screens: the Main Window and the Filter Pop-up Menu. In the Main Window are the Menu Bar, the State Panel, the Action Panel and the Message Panel. In the Filter Pop-up Menu are the toggle buttons that select the events to be filtered out. Below is a screen dump of the user interface. We will describe the user interface in more detail in the following sections.



Event Daemon Main Window

Event Filter Pop-up Menu

Figure 9: Event Daemon User Interface

5.2 Architecture

The ED is composed of 7 units: Receiver, Sender, Filter, User Interface, Converter, Queue and Permanent Storage. These units can be categorized by their functionality. The Sender and the

Receiver comprise the interclient communication interface with the CM. The User Interface provides the communication mechanism with the user. The Filter and Converter are "transition" tools, they check the events that are shipped across ED units doing filtering or conversion. And the Queue and Permanent Storage are storage pools in main memory and file system respectively.

The Receiver on the ED side is similar to that of the CM in some aspects. It decodes the event and executes the appropriate action according to the type of event. However, the format of the information received is completely different. Events are received synchronously in sets of three from the CM. The first event is an actual event, which is user or window-manager generated. The two following events are `ClientMessage` events that store time and name information. All this information is stored in a single node in the EdQueue. Since the ED stores information on the CM in the `EdInfo` structure, it is trivial to know whether an event is coming from the CM. To store the actual event it calls `EnQueue` and for the two trailers it calls `AddTime` and `AddName` respectively.

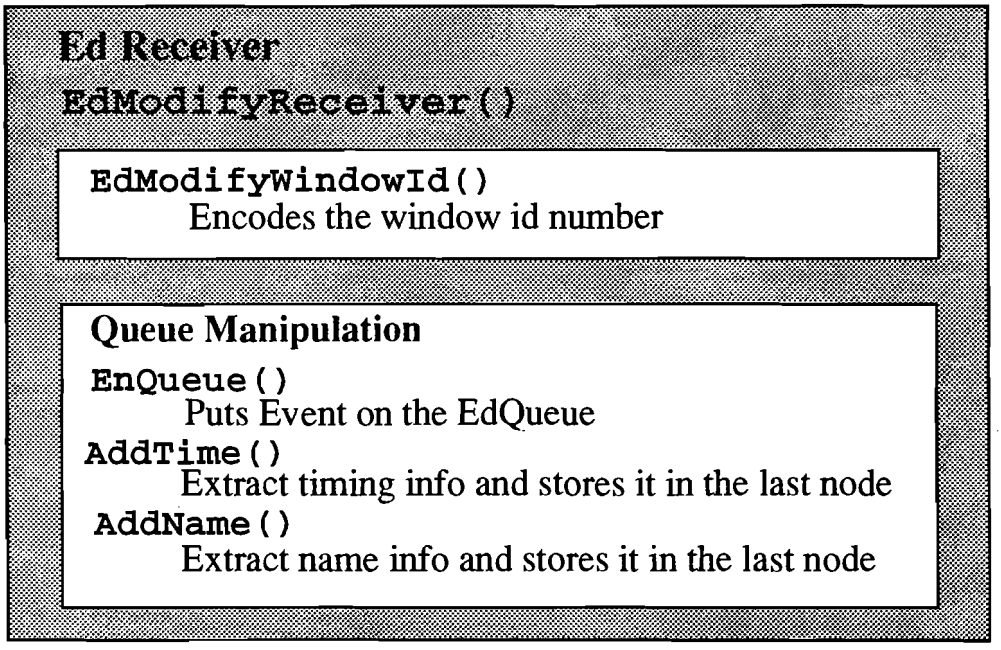


Figure 10: Components of `EdModifyReceiver()`

The Sender works differently from the one in the CM. Whereas in the CM, the Sender is traversed every time a loop is completed in the event loop, in the ED, the Sender is called asynchronously due to the event driven nature of the interface provided by the ED. Messages and events can be sent from the ED to the CM whenever the user decides to do so. For instance, the user can execute these actions by simply pushing a button in the State Panel or by pushing the "Send" button in the Action Panel. Two types of messages can be sent from the ED to the CM.

The first one is to cause a change of state in the CM. This is handled by the `EdSendMessage` routine. This routine will send a `ClientMessage` event containing a message indicating a change of state. The format of this message is explained in detail in section 6.3., *ClientMessage Event Structure Format*. The second type of message, is an actual event that was recorded and was originally user or window-manager generated. This type of message is executed by the routine `EdSendEventQueue`.

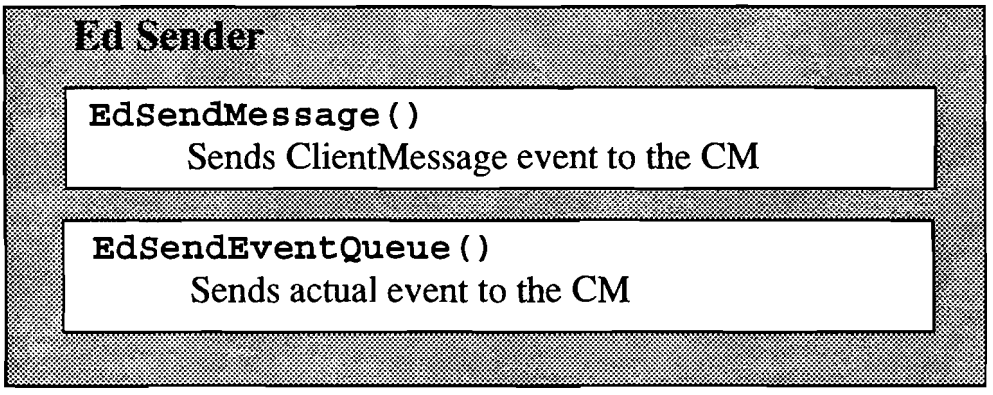


Figure 11: Components of the Event Daemon Sender

While the Receiver and the Sender provide the interface to communicate with the CM, the User Interface (UI) is the one that communicates with the user. The UI consists of 4 panels: The Menu Panel, Action Panel, State Panel and Message Panel as shown in *Figure 9*.

The Menu Panel is a menu bar with only one item: "File". By selecting this item, the menu will pop up a submenu with the items "New", "Save", "Retrieve" and "Quit". New invokes the callback `New_CB`, which deallocates the current EdQueue. Save invokes the callback `Save_CB`, which saves the current EdQueue to file using one of the Converter's routine `EdQueueToFile`. Retrieve invokes the `Retrieve_CB` callback which reads the default file[1] and creates a new EdQueue[2]. The last item in the submenu is Quit which exits the ED.

The Action Panel can be divided into two main parts: The selection part and queue/ message/filtering management part. The selection part consists of one button labelled "Select" and two text fields labelled "Window Id" and "Name". By pushing the Select button, the system will grab the cursor[3], then the user will select a target application with the left button of the mouse. The system will store window and name information from the selected application and

1. The default file for saving and retrieving is `output.xes`.
2. Retrieve also uses one of the Converter's routines: `EdFileToQueue`.
3. This process is done in routine `EdSelectWindow`.

will display it in the corresponding text fields[1].

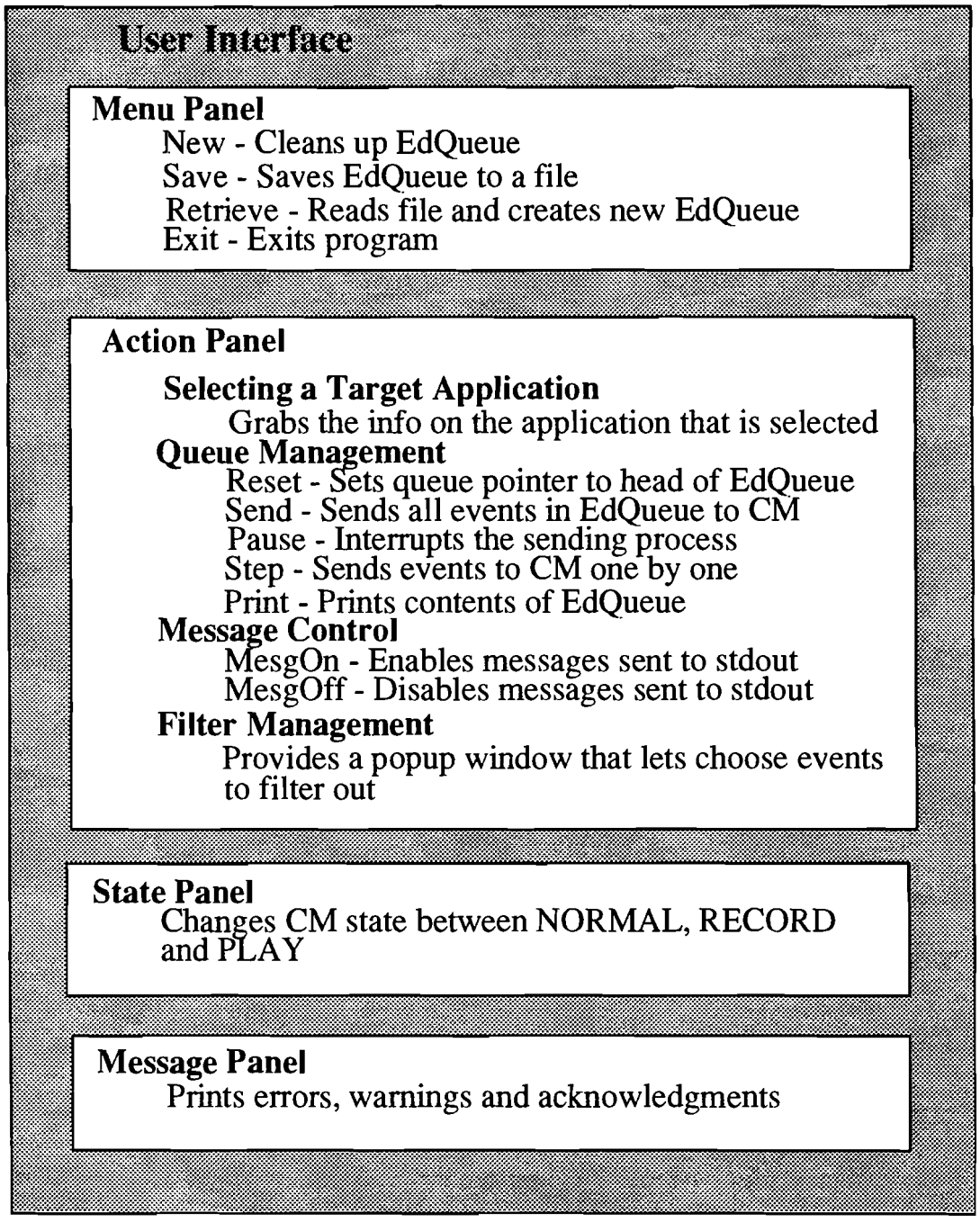


Figure 12: Components of the Event Daemon User Interface

1. `SelectPrintWin` and `SelectPrintName` are the routines in charge of display.

The queue management, message control and filtering functionality is provided through a set of buttons in the Action Panel: Reset, Send, Pause, Step, Print, MesgOn, MesgOff and Filter. The first five buttons deal with queue management. MesgOn and MesgOff turn on and off the message printing[1] to `stdout`. And Filter pops up the Filter Menu window.

The Reset button invokes the `QReset_CB` callback, which sets the current pointer to the EdQueue to the beginning of the queue. The Send button invokes the `QSend_CB` callback, which sets up the timeout function that will send the events to the CM in timely fashion. The timeout interval is determined from the information that was stored with every event that was captured[2]. This way we simulate the exact timing, that separated the original events. This playback of events will continue until the pointer reaches the end of the EdQueue or until the user pushes the Pause button. This will invoke the `QStop_CB` callback, which will remove the timeout. The Step button is similar to the Send button, but it only sends one event at a time and does not follow timing intervals. The last queue management button is the Print button, which invokes the `QPrint_CB` callback that will print the contents of the EdQueue to `stdout`.

MesgOn and MesgOff invoke `MessageOn_CB` and `MessageOff_CB` respectively. These callbacks will set the `print_flag` to either `True` or `False`. This flag is checked every time something is printed to `stdout`.

The Filter button will invoke `FilterPop_CB` which will pop up the Filter Menu window to allow the user choose the events to be filtered out. The filtering occurs both ways, from the CM to the ED and vice versa. In this way, the user can customize a test suite both at recording and playback time. Every toggle button in the Filter Menu window invokes the `ButtonToggle_CB` callback which sets a bit in a mask that stores information on the events to be filtered out. Then every time an event crosses the interclient communication channel, this mask is checked[3]. The "Done" button in the Filter Menu window pops down the window by calling the `FilterDown_CB` callback.

To the left of the Action Panel is the State Panel which is a set of three toggle buttons labelled: "Normal", "Record" and "Play". These buttons invoke the callbacks `Normal_CB`, `Record_CB` and `Play_CB`. These callback will send a message to the CM indicating a change of state. This protocol is described in more detail in section 6.3.1, *Changing CM States*.

The last panel in the User Interface is the Message Panel which lies below the State and Action panels. The Message Panels is a scrolled text widget that prints warnings, errors

---

1. Messages that are printed contain information about the events that are being sent and received across the interclient communication channel. These messages provide very detailed information and can be voluminous, thus they might slow down the recording or playback process. Therefore, turning messages off (by pushing the MesgOff" button) is recommended in order to achieve optimal performance.
2. During Record mode, every event is sent synchronously with time and name trailers.
3. For more detail on filtering, see section 6.4, *Event Filtering Scheme*.

acknowledgments and other information related to the status of the system. Old messages can be obtained by scrolling the text widget using the scrolling arrows. However, this text widget has limited buffer. All messages displayed in this panel are sent via the `MessagePrint` routine.

The transition tools of the ED are used when events are moved from one unit to another. They are the Filter and the Converter. The Filter stores its information in a mask called `filter_mask` which is set via the user interface as we mentioned before. Whenever an event comes or leaves the ED, the `filter_mask` is checked with the routine `CheckFilterEvent`.

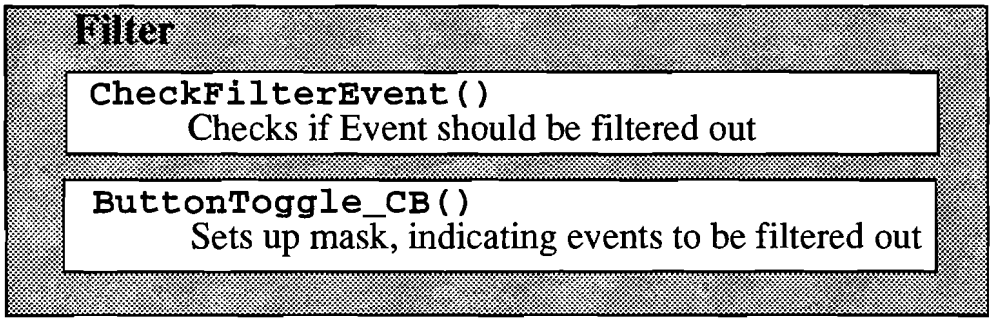


Figure 13: Components of the Filter

The other transition tool is the Converter which converts events from memory to file format and vice versa. The routines in the Converter (`EdQueueToFile` and `EdFileToQueue`) are used by save and retrieve option of the user interface.

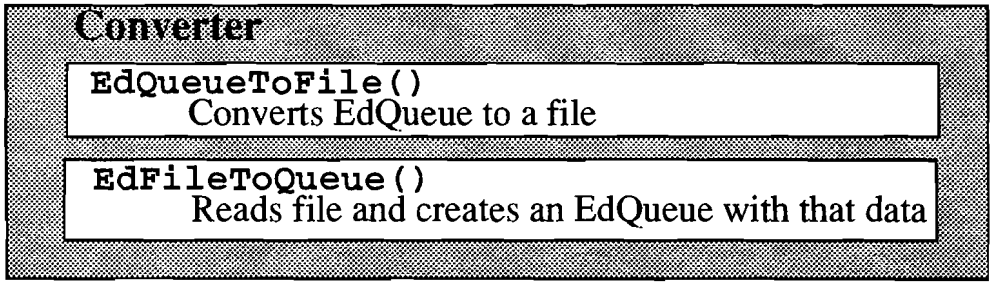


Figure 14: Components of the Converter

Finally, the Queue (or "EdQueue") and the Permanent Storage are passive repositories of events. The Queue is located in memory and has a FIFO structure, like any other queue. Each node in the queue contains an event, timing information and the name of the widget that originated the event. Permanent Storage is located in the file system. The file that stores the events is in ASCII format and can be even edited. However, additional tools should be built to manipulate this file, since direct editing can cause consistency errors. The file format is simple, all

the entries except the `display` and `window` fields are written out.

5.3 Data Flow

As we have seen the flow events from the ED to the CM can happen asynchronously since it is driven by the user interface. However, if the system is in Record mode, the process is consistent over time. Events coming into the ED are handled by the routine `EdModifyReceiver` and the process can described with the following illustration:

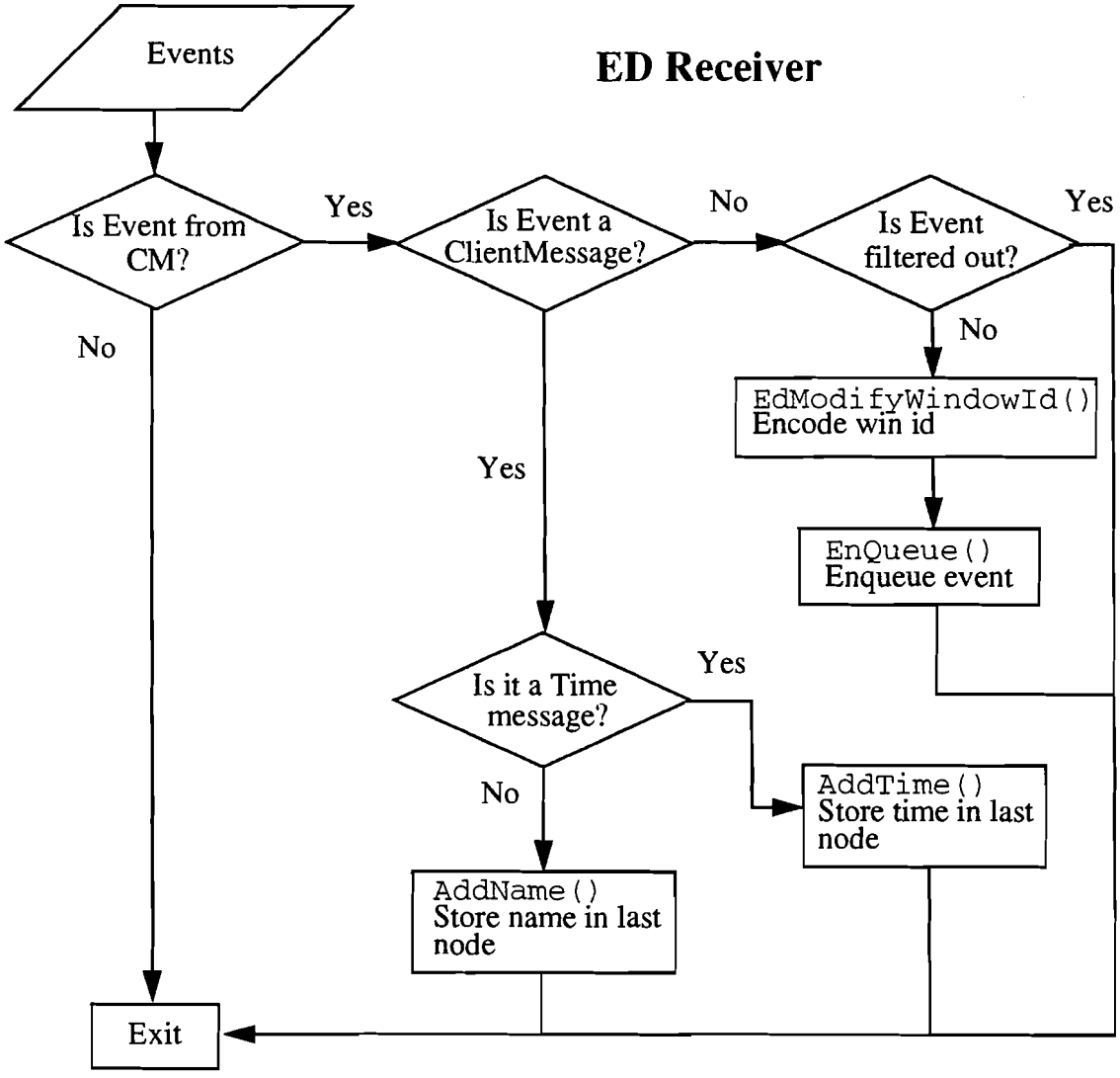


Figure 15: Data Flow Diagram of `EdModifyReceiver()`

Events are fed into the Receiver. First, it distinguishes the events that are coming from the CM and work only on these. The Receiver does not check the current state of the system, since that is left to the CM, in other words, the CM would not send any events if it is not in Record mode. Following the initial filtering, the Receiver checks whether the event is a `ClientMessage`. If it is not, it is a user or window-manager generated event, which will be filtered through the Filter, using the `CheckFilterEvent` routine. Following, the event's `window` field is encoded, before it is put on the EdQueue.

A `ClientMessage` coming from the CM would either be a time or a name message. The information in these messages is extracted and stored with the associated event. Time and name messages come as trailers following every event that comes from the CM.

The data flow of the other units of the ED has been already described in the previous section. They are mostly, two step processes started by some action in the user interface followed by a callback that invokes some routine in the Sender, the Filter or the Converter.

6 The Event Messaging Protocol

Event capture and replay would be a simple task if were to do all our capturing and storing in a single process. In that case, we just intercept all events and write them to a file. One way to do this is doing it in "batch mode", that is, we indicate at the beginning of run time whether we would like to capture or replay events. This is extremely inflexible and limits the user's actions. Ideally, we would like the user to interact with the capturing process, letting the user filter out unwanted events and replay sections of a test suite. However, due to the way user interface systems like X Window are built--based on a main event loop, user interaction and event capturing in the same process is complicated, if not impossible. In this case, it would be difficult to discern which events are meant to be captured and which are meant to interact with or control the capturing or replay process. If this distinction is not accomplished cleanly, the system might capture events that are meant to control the process, resulting in a chaotic situation when these events are replayed. This is the primary reason that made us separate the functionality of a capture and replay system into two X client processes and how they communicate is what we will describe in this chapter.

The two processes in XEMS communicate using interclient communication mechanisms provided by X. As we describe earlier, one of the processes is the target application, which has the Client Module (CM) attached to it. The other process is the Event Daemon (ED), which is in charge of managing the whole process of capturing and replaying events by providing a graphical user interface. In addition, the separation into two processes makes this system modular. Because of this modularity, XEMS can be reused for other kinds of systems that involve event capturing

like logging, demonstration and multiplexing.

The communication between the ED and the CM follows several protocols, that is, a common way to interpret messages by both sides. For this communication we must solve several issues like selecting the target application, identifying events that are coming from another process, sending messages and filtering unwanted events.

6.1 Target Application Selection Mechanism

The Target Application Selection Mechanism provides a way to establish initial communication between the ED and the CM. As a pre-requisite, the target application selected has to have a CM module installed with it[1]. The Selection Mechanism allows the user to point and click the target application the user wants to capture events from or replay events. The point and click feature is accomplished by grabbing the cursor and storing the information of the process selected. X provides a set of routines to handle this task[2]. Once the target application is selected, the information on this process, namely its toplevel window identifier number, is stored in the global structure `EdInfo`. The same information pertaining to the ED will be sent in the first change of state message that is sent to the CM[3]. Once the CM receives this message, it will have a handle for the ED, and thus can start sending messages to it. In this way, we have established the channel of communication between the two X client processes.

6.2 Window Identifier Number Encoding and Decoding

In order to identify "artificial" events that come from the ED, the window identifier number stored in the `window` field of every event structure stored in the ED side is encoded in a way that is recognized easily by the CM[4]. *Figure 16* describes the encoding and decoding process that the window identifier number goes through. A window identifier number is a 32-bit or 4-byte hexadecimal number that is assigned to every widget that is created. In other words, each widget has a unique window id associated with it, thus a specific widget can be keyed by its window id.

The window id has two parts, the high order bits (bits 16 to 31 from right to left) and the low order bits (bits 0 to 15). In the high order bits (bits 20 to 27), is a number that is common for all the widgets in the same process. This number identifies an X client and it is issued in increments of 0x400000. In the low order bits (bits 0 to 7) is another number that is unique to each widget. This number is consistent from instance to instance, thus allowing us to identify a particular widget. These numbers start at 0x10 (which is the id of toplevel widget for every X

1. See section 7.1, *Installation*, for more details.
2. The selection mechanism is implemented in `EdSelectWindow` which was obtained from `xev`, a public domain application available via ftp from `export.lcs.mit.edu` in the `/contrib` directory.
3. Actually the ED window id is sent with every change of state message.
4. The encoding generates a number that X would never produce.

client) and issued in increments of 0x1.

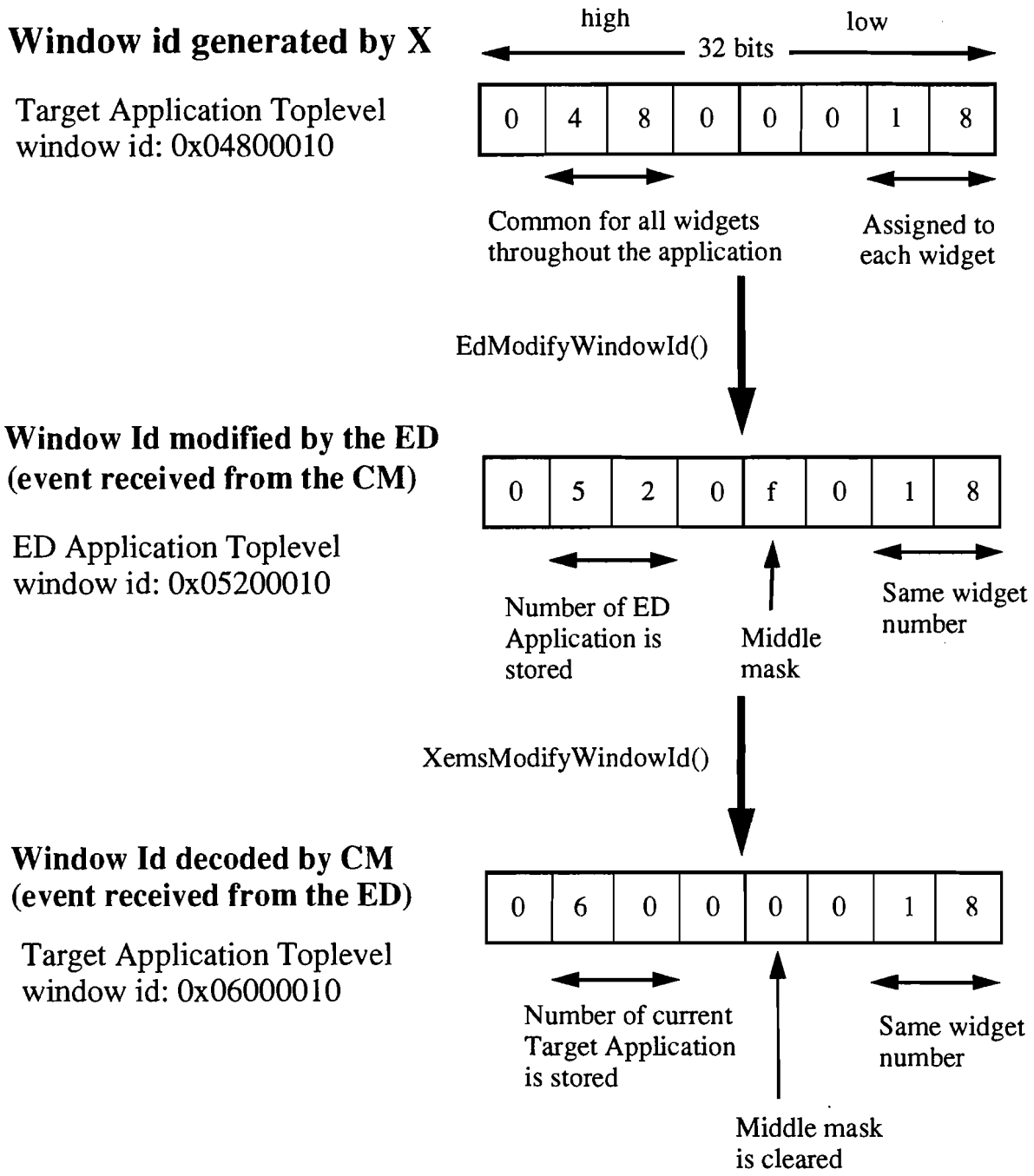


Figure 16: An example of Window Identifier Number Encoding and Decoding

In our example in *Figure 16*, the target application has issued the window id 0x04800018 for some widget. The window id for the toplevel widget for that X client will be 0x04800010. Notice that higher bits (0x0480) are common and the lower bits (0x0010 and 0x0018) are

different. Once an event containing the 0x04800018 window id is sent from the CM to the ED, it is encoded by `EdModifyWindowId`. The encoding process involves two steps: One on the higher bits and the other on the lower ones. On the higher bits, the number that identifies the client (0x0480 in our example) is overwritten by the number that identifies the ED (0x0520; 0x05200010 is the window id the toplevel of the ED). The other step is writing the number 0xf (or "middle mask" whose size is 4 bits) starting on bit 15. This number will mark the event as coming from the ED, since X would hardly produce this number[1]. The purpose of storing the ED's number on every event is to identify the event as coming from the ED. On the CM side, the Receiver checks that this number matches the higher order bits of the window id stored in the global structure `CmInfo`. There can be a case where the higher order bits of a stored event window id matches the higher order bits of a a new instance of an application. That is the purpose of the middle mask, it uniquely identifies and event as coming from the ED.

Following our example, once on the ED side the event can be stored to file or left in memory in the EdQueue. Before it is stored an event has to be encoded by `EdModifyWindowId`.This encoding ensures that the CM will never be in a confusing situation and will always know which events are being sent by the ED. Having done the encoding, the event is ready to be sent to the CM for replay. All events from the ED to the CM are sent to the toplevel widget of the CM[2].

Once in the CM, an event is recognized as coming from the ED by checking the high order bits and the middle mask. Then the CM proceeds to decode the window id using the routine `XemsModifyWindoId`. This routine will produce a window id that will resemble one generated by X, so that the widget dispatcher will never tell the difference. In this decoding process, the high order bits are overwritten with the CM's application number and the middle mask is cleared. The lowest byte will remain the same since this number will help on keying on the destination widget. Once the decoding is finished, the event is passed to `XtDispatchEvent` which will believe that the event was user or window-manager generated.

6.3 ClientMessage Event Structure Format

`ClientMessage` events are used for two purposes: To change the state of the system with a message from the ED to the CM and to send additional information associated to an event from the CM to the ED.

The `ClientMessage` structure has the following format:

```
typedef struct {
```

1. In order to generate this number, the application would have at least 2^{12}=4096 widgets, then again, we can accommodate our middle mask to take less space.
2. One of the parameters of `XSendEvent` is destination window id.

```
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window window;
    Atom message_type;
    int format;
    union {
        char b[20];
        short s[10];
        long l[5];
    } data;
} XClientMessageEvent;
```

We store our information in the `data` field of the `ClientMessage` event.

6.3.1 Changing CM States

A change of state is requested by the user through the user interface. The user can change among the Normal, Record and Play modes at any time. Every time, a change of state occurs, a `ClientMessage` event structure is sent from the ED to the CM indicating the appropriate change. The format of the message is depicted in the following illustration:

Changing State

Sent by: ED
Received by: CM
Created by: `EdCreateMessage()`

0	1	2	3	4	5...														19	bytes
"s"	d	x	x	x	x															

Byte 0: "s" for "state" (character format)
Byte 1: State to be changed to 0-NORMAL, 1-RECORD, 2-PLAY (short int format)
Bytes 2-5: ED Toplevel window id (hexadecimal long format)
Bytes 6-19: Unused

Figure 17: Format of a `ClientMessage` event containing a change of state message

The change of state message is created by the routine `EdCreateMessage`. It begins with the letter "s" stored as a character in byte 0. It is followed by byte 2, which contains the state. A "0" means "Normal", "1" "Record" and "2" means "Play". Bytes 2 to 5 contain the window id of the toplevel widget of the ED. This last information is used to establish the initial connection between the ED and the CM.

6.3.2 Storing Time and Name Information

Time and Name information is stored in two `ClientMessage` events that trail an actual event sent by the CM to the ED. Time information includes a long integer containing the system time in milliseconds of the moment that the event was captured. Time stamps of adjacent events can then be compared to compute the interval between them at playback time. The time stamp occupies the first long integer of the `data` field in the `ClientMessage` event.

The Name trailer follows the Time trailer. The name is a string whose length is no longer than 20 bytes and is null terminated. This string contains the name of the widget that originated the event (on the target application). The purpose of storing the name is to uniquely identify the widgets. Similar to the window id information, names can be used to key on the widgets of an application. However, X does not provide routines to map widget names to window identifier numbers. Such mechanism does not rely on the order of creation--like the window identifier numbers do--and can be used as another way to dispatch the events to the widgets that originated them.

Timing Information

Sent by: CM
Received by: ED
Created by: `XemsCreateTimeTrailer()`

0	1	2	3	4	longwords (32 bits)
dddddddd					

Longword 0: System time in milliseconds (decimal format, long int)
Longwords 1-4: Unused

Name Information

Sent by: CM
Received by: ED
Created by: `XemsCreateNameTrailer()`

0	1	2	3	4	5...														19	bytes
w	i	d	g	e	t	n	a	m	e	\0										

Bytes 0-19: Widget Name string, null terminated (char format)

Figure 18: Format of Time and Name trailer messages

6.4 Event Filtering Scheme

The Filtering Scheme has two periods: The Setup period and the Check period. In the Setup period the user decides which events to filter out. The user can do this through the ED user interface, namely the Filter Menu Popup window. The toggle buttons in this menu will activate the `ButtonToggle_CB` callback which will set the bits in the filter mask. This filter mask consist of two long integers[1] arranged in an array. A bit is assigned for each type of event. Whenever this bit is set, it means that this event should be filtered out. The mask is set up by writing the bit corresponding to the type of event.

In the Setup period, we set the corresponding bit without altering the rest of the bits. Let us look at the example in *Figure 19*:

Filter Mask

long filter_mask[0]:	Events 0-30 (inclusive)
long filter_mask[1]:	Events 31-32

Given:

filter_mask[0]:	0110
Event type:	3
state:	1

Operations:

filter_mask[0]	0110
~(1L << type)	0111
bitwise AND	0110
(state << type)	1000
bitwise OR	1110

Figure 19: Example of a Filter Setup operation

In the example `filter_mask[0]` has bits 1 and 2 set[2] (from right to left) which means that events of type 1 and 2 should be filtered out. To check this against the mask, we find the position of this type of event (`1L<<type`). In this case `1<<3`, which results in `1000`. We then negate the entire expression (`~(1L<<type)`) to obtain `0111`. Next we perform a bitwise AND

1. Since there are more than 31 types of events and 0 is reserved, we need more than 32 bits.
2. We are not including the other 28 bits for brevity.

between the `filter_mask[0]` and the negated expression: `0110 AND 0111 = 0110`. This operation will clear the bit corresponding to type 3 and will leave the rest of the bits untouched. Finally, we perform a bitwise OR between the previous result and the state of the button stored in the position corresponding to type 2. Assuming that the state of the toggle button for event type 3 is pushed (has value 1), we have: `0110 OR 1000 = 1110`. `1110` is the new value of the of the `filter_mask[0]` and it denotes that events of type 3 should be filtered out.

In the Check period, we find out whether the bit in the mask corresponding to the type of event is set. We can accomplish this by doing a bitwise AND between a mask containing the position of the event and the filter mask. Thus, the operation will be: `(1L<<type) AND filter_mask`. A return value of `1` will mean that the event should be filtered out and a value of `0` would mean the contrary.

7 User's Guide

7.1 Installation

To set up the system both applications should be running: the target application and the Event Daemon (ED). Previously the target application must be compiled with `XemsMainLoop` as the main event loop or linked with a modified Xt library (`libXt.a`) provided by XEMS. The way we modified `libXt.a` is by replacing `XtAppMainLoop` with the contents of `XemsMainLoop`. `XtAppMainLoop` is in located in the `Event.o` file of `libXt.a`.

7.2 Selecting a Target Application

To start capture or playback, the user must first establish the communication between the target application and the ED. To do so, select a target application by pushing the "Select" button in the Action Panel. The cursor bitmap will turn into a crosshair. Then proceed to point and click the application of your choice (that has been previously installed). The window identifier number and name of the application will be displayed on the text fields labelled "Window Id" and "Name" in the Action Panel. From then on, you are ready to change state and start capture or replay.

7.3 Changing State

In order to capture, the system must be in "Record" mode. Once you have selected the target application, you can switch to Record mode by pushing the toggle button labelled "Record" in the State Panel. Messages confirming your choice will appear on the Message Panel. Similarly, you can switch to "Normal" mode in which no recording or playback is done or to "Play" mode in

which events are sent from the Event Daemon (ED) to the target application.

7.4 Queue Management

In "Play" mode, a number of functions are available to control the events that have been stored in the queue on the ED side. The "Reset" button sets the pointer to the head of the queue. The "Send" button will send all the contents of the queue to the target application in a timely fashion, following the same time intervals between events when they were captured. This process can be interrupted by pushing the "Pause" button. The "Step" button is similar to "Send", but only sends one event at a time and does not follow timing. Finally, the "Print" button prints the contents of the current queue on the shell screen that launched the ED.

7.5 Turning Messages On and Off

Messages that are printed on the shell screen can be turned on and off by pushing the "MesgOn" and "MesgOff" buttons. This messages contain detailed information on the events that are being sent and can slow down the recording or playback process.

7.6 Filtering Events

Push the "Filter" button in the Action Panel and the Filter Menu window will pop up. It is a menu of toggle buttons, one for every type of event. To filter out a type of event push the corresponding toggle button. The filter work both ways, when events are captured and when they are replayed. All events are allowed to pass by default.

7.7 Saving and Retrieving Events

Once you have captured a set of events, they are stored in memory on the ED. To save them for later replay go to the Menu Panel and select the "File" item on the menu bar. This will bring a submenu, containing the "Save" item. If this item is selected, all the events will be written to a file called "`output.xes`". To retrieve from this file, select the "Retrieve" option on the submenu and the current contents of the queue will be overwritten by the contents of the file.

7.8 Resetting and Quitting

To clear the contents of the current queue push the "New" option on the submenu that is brought up by the "File" item on the menu bar. To exit the program, select the "Quit" option.

7.9 Scenarios

To select a target application: see section 7.2, *Selecting a Target Application.*

To start recording: Go to the State Panel and select the "Record" toggle button. Now the target application is ready to record. Proceed to interact with the target application. All the events that are generated are being recorded, unless you choose to filter some of them out. Messages

about these events will be displayed on the shell screen that launched the ED. To turn them off, push the "MesgOff" button in the Action Panel. The turn them back on, push the "MesgOn" button.

To stop recording: Go to the State Panel of the ED and push the "Normal" toggle button. The target application will stop sending events.

To save your events: Now you can save the events you just captured by going to the Menu Panel and selecting the "Save" option on the submenu that is brought up by the "File" item on the menu bar. Once you saved your events, you can exit both the ED and the target application.

To retrieve and playback: Start up both applications, the ED and the target application. Establish the connection between them by selecting the target application[1]. Retrieve your events by selecting the "Retrieve" option on the submenu of the "File" item on the menu bar. Your events are now loaded on the EdQueue. Go to the State Panel and select "Play" mode. Now the target application is ready to receive events. You can see the events replayed exactly the way they were generated by pushing the "Send" button. To interrupt this process press the "Pause" button. Alternatively, you can watch your events being replayed one by one by pushing the "Step" button consecutively.

To replay again: Reset the queue by pushing the "Reset" button in the Action Panel. If you push "Send" or "Step" now, events will be reproduced from the beginning. To see the contents of the queue: Push the "Print" button.

8 Conclusions and Outlook

XEMS is a client-server system that allows the duplication, storage and manipulation of user generated events. Its interclient communication mechanism is built exclusively with Xlib calls and its user interface is built with the Motif library; thus, XEMS is portable across platforms and toolkits. A target application that interacts with XEMS, can be built with the Athena, Motif, OpenLook or any other Xt toolkit. In summary, XEMS provides an interactive, modular, open and flexible framework to capture and replay events. For this project, we have implemented a regression tester using XEMS.

In addition to regression testing, XEMS can be used for a variety of purposes like session logging, local and remote demonstrations, and multiplexing. Logging is one of initial steps for regression testing, since we need to capture a test suit from user input. Local demonstration is accomplished by retrieving events from permanent storage and playing it back. Remote demonstration involves at least two instances of the same program, one being displayed at a

1. See section 7.2, *Selecting a Target Application.*

remote machine. Events generated on one instance of the program will be reproduced on the other one, and vice versa. Multiplexing is similar to remote demonstration, but it can involve more than two instances of the same program. In this case, the broadcasting of events is more complex, and it should have a specialized protocol to regulate event traffic. These are the several applications for which XEMS might be useful.

The proposal of XEMS has been warmly received by colleagues across the Internet. Its portability and simplicity seem to be its strongest feature.

9 References

[1] Boris Beizer. *Software Testing Techniques.* Van Nostrand Rheinhold, 1990.

[2] Alan Jamison. *The XTrap Architecture.* Digital Equipment Corp., 1991.

[3] Oliver Jones. *Introduction to the X Window System.* Prentice Hall, 1989.

[4] Montly L. Hammontree, Jeffrey J. Hendrickson and Billy W. Hensley. "Integrated Data Capture and Analysis Tool for Research and Testing on Graphical User Interfaces" in *Proceedings of CHI 1992.* ACM, 1992.

[5] Dan Heller. *Motif Programming Manual, OSF/Motif version 1.1.* O'Reilly & Associates, Inc., September 1991.

[6] Brian W. Kernighan and Denis M. Ritchie. *The C Programming Language.* Prentice Hall, 1988.

[7] Adrian Nye. *Xlib Programming Manual.* O'Reilly & Associates Inc., October 1990.

[8] Adrian Nye, ed. *Xlib Reference Manual.* O'Reilly & Associates, Inc., October 1990.

[9] Adrian Nye and Tim O'Reilly. *X Toolkit Intrinsics Programming Manual.* O'Reilly & Associated, Inc., September 1990.

[10] R. W. Scheifler and J. Gettys. X *Window System, 2nd ed.*, Digital Press, 1990.

[11] UniSoft Group Limited. *Xlib and X Protocol Test Suite.* July 1992.

[12] Young, Douglas A. T*he X Window System, Programming and Applications with Xt, OSF/Motif Edition.* Prentice Hall, 1990.

Public Domain Software:

[13] xev - An event diagnostics package.

[14] xrecplay - A capture and replay package that uses XTest [11].

[15] xsendevent - A command line event sender.

These packages were obtained via `ftp` from export.lcs.mit.edu (`/contrib` directory).

10 Appendix

These are the files that comprise XEMS:

For the ED:

```
makefile
ed.h            ed.c
file.h          file.c
queue.h         queue.c
ui.h            ui.c
                uiAction.c
                uiFilter.c
                uiMenu.c
                uiMessage.c
                uiState.c
```

For the CM:

```
makefile
XemsEvent.h     XemsEvent.c
```

Utilities:

```
util.h          util.c
```

Source code listings follow.

```
# use ANSI C Compiler
CC = /usr/lang/acc

NAME     = xtmain
BINARIES = Event.o Intrinsic.o XemsEvent.o util.o
CFLAGS   = -I./natasha/sys2/X11R5/mit/lib/Xt -I./natasha/sys2/X11R5/mit -g
DEFINES  = -DINCLUDE_ALLOCA_H
SHLIB    = libXt.a

# Define a rule for converting .c files into o. files
.c.o:      ; ${CC} -c ${CFLAGS} $(DEFINES) $*.c


${NAME} : ${BINARIES}
        ${CC} -c ${CFLAGS} $(DEFINES) ${BINARIES}

xemslib:
        ar rav Event.o ${SHLIB} Event.o
        ar rav Intrinsic.o ${SHLIB} Intrinsic.o
        ar rav XemsEvent.o ${SHLIB} XemsEvent.o
        ar rav util.o ${SHLIB} util.o
        ranlib ${SHLIB}

```

```
/*********************************************************************
 * XemsEvent.h
 *********************************************************************
#ifndef _XEMS_EVENT_H_
#define _XEMS_EVENT_H_

#define MAX_WIN_TABLE_ENTRIES 32
#define ALLEVENTMASK KeyPressMask|KeyReleaseMask|ButtonPressMask|ButtonReleaseMask|EnterWindowMask|LeaveWindowMask|PointerMotionMask|PointerMotionHintMask|Button1MotionMask|Button2MotionMask|Button3MotionMask|Button4MotionMask|Button5MotionMask|ButtonMotionMask|KeymapStateMask|ExposureMask|VisibilityChangeMask|StructureNotifyMask|ResizeRedirectMask|SubstructureNotifyMask|SubstructureRedirectMask|FocusChangeMask|PropertyChangeMask|ColormapChangeMask|OwnerGrabButtonMask

#define  HIGH_MASK    0xffff0000        /* Used filter win id's. The high */
                                        /* order bits are common to all */
                                        /* widgets in the same application */
                                        /* and the low order bits identify */
                                        /* each widget */
#define  MIDDLE_MASK  0x0000f000        /* We use these bits to "codify" */
                                        /* a win id; however we assume */
                                        /* that there will be less than */
                                        /* 0xFFF (4095) widgets per appl. */

#include <X11/Xlib.h>


/* Global Window ID Table (Not used in this version) */
/*
struct
{
  int last_entry;
  Window array[MAX_WIN_TABLE_ENTRIES];
} WinTable;
*/

/* Enumeration of Client Module (CM) states */
typedef enum {NORMAL, RECORD, PLAY} CMState;

/* Struct that stores global Info */
typedef struct
{
  Bool init;             /* Boolean: initialized or not */
  Window  ed_window;    /* ED's window id */
  Display *ed_display; /* ED's display ptr */
  Window  cm_window;    /* CM's (App) window id */
  Display *cm_display; /* CM's (App) display ptr */
  CMState state;        /* CM's state: normal, record, play */
  int x, y;             /* CM's (App) top left corner */
  unsigned int width, height;  /* CM's (App) width and height */

} GlobalInfo;


/*********************************************************************
 * Prototypes
 *********************************************************************/
/* Main loop */
void XemsMainLoop();        /* XEMS Main loop */

/* Messaging Routines */
void XemsModifyEvent();     /* Plugs into Mainloop (setup/rec/send) */
void XemsModifySetup();     /* Sets up initial state */
void XemsModifySender();    /* Sends events to ED */
void XemsModifyReceiver(); /* Receives messages from ED */

Bool XemsDetectFromEd();         /* Checks if event came from ED */
void XemsModifyWindowId();       /* Decodes scrambled window id */

void XemsModifyReceiverClientMessage(); /* Handles Client Message */
                                        /* events from ED*/
void XemsModifyReceiverConfigureNotify(); /* Handles win manager */
                                          /* generated post events */
void XemsModifyReceiverMapNotify();     /* Similar to above but for */
void XemsModifyReceiverUnmapNotify();   /* map/unmap events */

void   XemsSendEvent();          /* Sends event to ED */
Window XemsFindWindow();         /* Finds id of window with given name */
void   XemsCreateNameTrailer(); /* Creates Client Message, storing name of */
                                 /* widget */
void   XemsCreateTimeTrailer(); /* Creates Client Message, storing system */
                                 /* time */


/* Hash Table Routines */
void XemsInitWinTable();         /* Not used for this version */
void XemsInsertWinTable();       /* but might be useful in the future */
void XemsPrintWinTable();
Window XemsReturnEntryWinTable();


#endif
```

```
1   /*************************************************************************
2    * XemsEvent.c
3    * by Eddy Wong
4    * March 28, 1993
5    *************************************************************************/
6   #include <stdio.h>
7
8   #include <Xll/Intrinsic.h>
9
10  #include "XemsEvent.h"
11  #include "util.h"
12
13  GlobalInfo cm_info = (False,0,0,NORMAL); /* Global struct that keeps info */
14  GlobalInfo *CmInfo = &cm_info;           /* about win, disp, state of the */
15                                           /* CM */
16  static int config_notify_count = 0;      /* Counter of configure notify */
17                                           /* events */
18
19  /*************************************************************************
20   * void XtAppMainLoop(app)
21   * Input:       app - Application Context
22   * Output:      None
23   * Algo:        Infinite loop, intercept events betw. XtAppNextEvent and
24   *              XtDispatchEvent.
25   * Side Effects:        If the event is a client message coming from the
26   *                      ED, the action associated to that message will
27   *                      be executed and the event struct will be blanked
28   *                      out by XemsModifyEvent (there will be no effects
29   *                      if the the event goes thru XtDispatchEvent)
30   *************************************************************************/
31
32  /* Original in Event.c */
33  void XtAppMainLoop(app)
34  XtAppContext app;
35
36  {
37      XEvent event;
38
39      for (;;) {
40          XtAppNextEvent(app, &event);     /* Get event from queue */
41          XemsModifyEvent(&event);         /* Filter events, execute actions */
42                                           /* send events to ED */
43          XtDispatchEvent(&event);         /* Distribute events to widgets */
44      }
45  }
46  /* Original in Event.c */
47
48
49  /*************************************************************************
50   * void XemsMainLoop(app)
51   * Input:       app - Application Context
52   * Output:      None
53   * Algo:        Infinite loop, intercept events betw. XtAppNextEvent and
54   *              XtDispatchEvent.
55   * Side Effects:        If the event is a client message coming from the
56   *                      ED, the action associated to that message will
57   *                      be executed and the event struct will be blanked
58   *                      out by XemsModifyEvent (there will be no effects
59   *                      if the the event goes thru XtDispatchEvent)
60   *************************************************************************/
61  void XemsMainLoop(app)
62  XtAppContext app;
63  {
64      XEvent event;
65
66      for (;;) {
67          XtAppNextEvent(app, &event);     /* Get event from queue */
68          XemsModifyEvent(&event);         /* Filter events, execute actions */
69                                           /* send events to ED */
70          XtDispatchEvent(&event);         /* Distribute events to widgets */
71      }
72  }
73
74
75  /*************************************************************************
76   * void XemsModifyEvent(event)
77   * Input:       event - Event structure
78   * Output:      None
79   * Algo:        Setup, receive and send events.
80   * Side Effects:        Nullify client message events coming from the ED.
81   *************************************************************************/
82  void XemsModifyEvent(event)
83  XEvent *event;
84  {
85    XemsModifySetup(event);
86    XemsModifyReceiver(event);
87    XemsModifySender(event);
88  }
89
90  /*************************************************************************
91   * void XemsModifySetup(event)
92   * Input:       event- Event structure
93   * Output:      None
94   * Algo:        Store win and disp info, setup flag
95   * Side Effects:        Sets flag so that boy of routine is not repeated
96   *************************************************************************/
97  void XemsModifySetup(event)
98  XEvent *event;
99  {
100   if (CmInfo->init == False)
101     {
102       CmInfo->init = True;
103       CmInfo->cm_window = event->xany.window;
104       CmInfo->cm_display = event->xany.display;
105
106       /*
107       printf("Cm init %d\n", CmInfo->init);
108       printf("Cm window %x\n", CmInfo->cm_window);
109       printf("Ed window %x\n", CmInfo->ed_window);
110       printf("Cm state %d\n", CmInfo->state);
111       */
112     }
113
114 }
115
116 /*************************************************************************
117  * void XemsModifySender(event)
118  * Input:       event - Event Structure
119  * Output:      Prints informational messages about event
120  * Algo:        Filters events and sends them to ED.
121  *              Check if event is NOT from ED, if so, check if it's
122  *              a configure notify event, if so, plug in info on
123  *              top left corner (request this info via X calls).
124  *              Be careful not to this on the first two conf notify
125  *              events, because, these come from the win manager,
126  *              and setup events (doing X calls i.e. XGetGeometry
127  *              for these would cause an X error). Finally, if
128  *              the CM is in RECORDING state, send the event to the ED.
```

```
 * Side Effects:        Modifies conf notify events (resize events),
 *                      since these don't include info of the top left
 *                      corner of the application window.
 ***********************************************************************/
void XemsModifySender(event)
XEvent *event;
{
  Window root, junkwin;                    /* window ids */
  int x, y, rx, ry;                        /* window coords */
  unsigned int width, height, bw, depth; /* window measurements */
  XWindowAttributes wattr;

  if ((event->type != 0) || (!XemsDetectFromEd(event)))
    {  /* If event is valid and NOT from ED */

      /* If it's an original move/resize event */
      if (event->type == ConfigureNotify)
        {
          if (((event->xconfigure.window & HIGH_MASK) ==
               (CmInfo->cm_window & HIGH_MASK)) &&
              ((event->xconfigure.window & MIDDLE_MASK) == 0))
            { /* if comes from the CM window and middle_mask is not set */

              /* Two config_notifies are issued when widgets */
              if (config_notify_count > 1)
                {              /* are realized, so skip those */
                  XGetGeometry(CmInfo->cm_display, event->xconfigure.window,
                               &root, &x, &y, &width, &height, &bw, &depth);
                  XGetWindowAttributes(CmInfo->cm_display,
                                       event->xconfigure.window,
                                       &wattr);
                  XTranslateCoordinates(CmInfo->cm_display,
                                        event->xconfigure.window,
                                        wattr.root,
                                        wattr.border_width,
                                        wattr.border_width,
                                        &rx, &ry, &junkwin);
                  /* This one gets the actual top left corner position!!! */
                  /*
                  printf("---> rx ry %d %d\n", rx, ry);
                  printf("---> w h %d %d\n", width,height);
                  */
                }
              config_notify_count++;

              if (event->xany.send_event == False) /* Resize event */
                {
                  /* Resize events don't include top-left corner info */
                  event->xconfigure.x = rx;
                  event->xconfigure.y = ry;
                }
              printf("******** Repeat **********\n");
              PrintEvent(event);
              printf("*************************\n");
            }
        }

      if ((CmInfo->state == RECORD))
        /* if event originated by user and we're in RECORD state */
        {
          printf("+++++++++++++++\n");
          if (CmInfo->ed_window)     /* if Event Daemon is alive send */
            XemsSendEvent(event);  /* Send Events to Event Daemon */
        }
    }
}

/***********************************************************************
 * void XemsModifyReceiver(event)
 * Input:        event - Event Structure
 * Output:       Information (about events) messages
 * Algo:         This routine is in charge of receiving messages from
 *               the ED. It checks if events are coming from the ED first,
 *               then it decodes the win id number, so that we can extract info
 *               about the widget that originated this event. Then it must
 *               respond according to the kind of message received. If it's
 *               a ClientMessage we call the ClientMessage handler. If
 *               it's an event and we are not in PLAY state, we dispose
 *               the event. Finally it can be a window-manager-generated
 *               event, like ConfigureNotify, Map/UnmapNotify, if so
 *               we call the appropriate routines, which in turn
 *               will extract info from the event struct and make
 *               the appropriate X calls to the window manager.
 * Side Effects:         If not in PLAY state, it will dispose the
 *                       event (set the type to 0, whic means that
 *                       it is an error even, for X)
 ***********************************************************************/
void XemsModifyReceiver(event)
XEvent *event;
{
  if (event->xany.send_event == True)
    {
      if (XemsDetectFromEd(event)) /* If event from ED */
        {
          *
          printf("**********Original*********\n");
          PrintEvent(event);
          printf("*************************\n");
          */

          /* copy 1st and 2nd high order bytes */
          XemsModifyWindowId(event);  /* decode win id # */
          /*
          printf("**********Changed*********\n");
          PrintEvent(event);
          printf("*************************\n");
          */

          if (event->type == ClientMessage)
            XemsModifyReceiverClientMessage(event);

          if ( (CmInfo->state == RECORD) ||
               (CmInfo->state == NORMAL))
            {  /* if state other than PLAY, don't pass any events from ED */
              printf("Dispose event\n");
              event->type = 0;
            }
          else
            {
              switch(event->type) /* window-mgr generated events */
                {
                case ConfigureNotify:
                  XemsModifyReceiverConfigureNotify(event);
                  break;
                case MapNotify:
                  XemsModifyReceiverMapNotify(event);
                  break;
                case UnmapNotify:
```

```
                    XemsModifyReceiverUnmapNotify(event);
                    break;
                  }
            }
        }
    }
}


/*********************************************************************
 * Bool XemsDetectFromEd(event)
 * Input:        event - Event Structure
 * Output:       True: from ED, False: Not from ED
 * Algo:         Checks the encoding in the win id #. If bits 12-15 (from
 *               right to left) are all set, that means that it comes
 *               from the ED. The ED encodes all win ids that it stores.
 *               To check this we just do a bitwise AND at the 12-15 bit
 *               locations, which are endoded in the MIDDLE_MASK mask.
 * Side Effects: None
 *********************************************************************/
Bool XemsDetectFromEd(event)
XEvent *event;
{
  switch(event->type)
    {
    case CirculateNotify:
      return (event->xcirculate.window & MIDDLE_MASK);
      break;
    case CirculateRequest:
      return (event->xcirculaterequest.window & MIDDLE_MASK);
      break;
    case ConfigureNotify:
      return (event->xconfigure.window & MIDDLE_MASK);
      break;
    case ConfigureRequest:
      return (event->xconfigurerequest.window & MIDDLE_MASK);
      break;
    case CreateNotify:
      return (event->xcreatewindow.window & MIDDLE_MASK);
      break;
    case DestroyNotify:
      return (event->xdestroywindow.window & MIDDLE_MASK);
      break;
    case GravityNotify:
      return (event->xgravity.window & MIDDLE_MASK);
      break;
    case MapNotify:
      return (event->xmap.window & MIDDLE_MASK);
      break;
    case UnmapNotify:
      return (event->xunmap.window & MIDDLE_MASK);
      break;
    case MapRequest:
      return (event->xmaprequest.window & MIDDLE_MASK);
      break;
    case ReparentNotify:
      return (event->xreparent.window & MIDDLE_MASK);
      break;
    default:
      return (event->xany.window & MIDDLE_MASK);
    }
}


/*********************************************************************
 * void XemsModifyWindowId(event)
 * Input:        event - Event structure
 * Output:       None
 * Algo:         Modifies the encodes window id (encoded by the ED), so that
 *               the widget dispatcher can recognize it. We have to
 *               set the higher 2 bytes of the event win id to that of
 *               the current application's win id (stored in
 *               CmInfo->cm_window) and reset (put zeros) in locations
 *               12-15 (right to left). This last one uniquely identifies
 *               events as coming from the ED.
 * Side Effects:        Sets the event's win id to a "workable" one,
 *                      that is, as if if was created by user input.
 *                      After this decoding, there is no way to know that
 *                      the event was produced by another instance of the
 *                      the application or that the event came from the
 *                      ED.
 *********************************************************************/
void XemsModifyWindowId(event)
XEvent *event;
{
  switch(event->type)
    {
    case CirculateNotify:
      /* Copy highest order 2 bytes, these bytes are common within the app */
      bcopy(&(CmInfo->cm_window), &(event->xcirculate.window),2);
      /* Reset (set to 0), bits 12-15, this is an encoding from the ED */
      event->xcirculate.window ^= MIDDLE_MASK;
      break;
    case CirculateRequest:
      bcopy(&(CmInfo->cm_window), &(event->xcirculaterequest.window),2);
      event->xcirculaterequest.window ^= MIDDLE_MASK;
      break;
    case ConfigureNotify:
      bcopy(&(CmInfo->cm_window), &(event->xconfigure.window),2);
      event->xconfigure.window ^= MIDDLE_MASK;
      break;
    case ConfigureRequest:
      bcopy(&(CmInfo->cm_window), &(event->xconfigurerequest.window),2);
      event->xconfigurerequest.window ^= MIDDLE_MASK;
      break;
    case CreateNotify:
      bcopy(&(CmInfo->cm_window), &(event->xcreatewindow.window),2);
      event->xcreatewindow.window ^= MIDDLE_MASK;
      break;
    case DestroyNotify:
      bcopy(&(CmInfo->cm_window), &(event->xdestroywindow.window),2);
      event->xdestroywindow.window ^= MIDDLE_MASK;
      break;
    case GravityNotify:
      bcopy(&(CmInfo->cm_window), &(event->xgravity.window),2);
      event->xgravity.window ^= MIDDLE_MASK;
      break;
    case MapNotify:
      bcopy(&(CmInfo->cm_window), &(event->xmap.window),2);
      event->xmap.window ^= MIDDLE_MASK;
      break;
    case UnmapNotify:
      bcopy(&(CmInfo->cm_window), &(event->xunmap.window),2);
      event->xunmap.window ^= MIDDLE_MASK;
      break;
    case MapRequest:
      bcopy(&(CmInfo->cm_window), &(event->xmaprequest.window),2);
      event->xmaprequest.window ^= MIDDLE_MASK;
```

```
385         break;
386       case ReparentNotify:
387         bcopy(&(CmInfo->cm_window), &(event->xreparent.window),2);
388         event->xreparent.window ^= MIDDLE_MASK;
389         break;
390       default:
391         bcopy(&(CmInfo->cm_window), &(event->xany.window),2);
392         event->xany.window ^= MIDDLE_MASK;
393       }
394   }
395
396
397   /*********************************************************************
398    * void XemsModifyReceiverClientMessage(event)
399    * Input:        event - Event structure
400    * Output:       Print info messages
401    * Algo:         Handle ClientMessage events. ClientMessage events
402    *               received would most probably signify a change of state.
403    *               A change-of-state ClientMessage event is encoded
404    *               with the letter 's' (1st byte), followed for by the state code
405    *               0: NORMAL, 1: RECORD, 2: PLAY (2nd byte) and the ED's
406    *               win id (3rd to 6th bytes)
407    * Side Effects:         Changes state of CM system.
408    *********************************************************************/
409   void XemsModifyReceiverClientMessage(event)
410   XEvent *event;
411   {
412     /* Change state if received any messages from ed */
413     if (event->xclient.data.b[0] == 's')
414       {
415         CmInfo->state = event->xclient.data.b[1]; /* store state code */
416                                                   /* change state actually */
417         printf("State %d\n",event->xclient.data.b[1]);
418         if (CmInfo->ed_window == 0)  /* copy ED window id from msg */
419           {
420             bcopy(&(event->xclient.data.b[2]),&(CmInfo->ed_window),4);
421             printf("Ed Window is %x\n",CmInfo->ed_window);
422           }
423       }
424     else
425       if (event->xclient.data.b[0] == 'n') /* ClientMessage with name of */
426         {                                  /* widget, not used in this version */
427           printf("string: %s\n",&(event->xclient.data.b[2]));
428         }
429
430     switch (CmInfo->state)  /* Print new state info */
431       {
432       case NORMAL:
433         printf("State: Normal\n");
434         break;
435       case RECORD:
436         printf("State: Record\n");
437             break;
438       case PLAY:
439         printf("State: Play\n");
440         break;
441       }
442   }
443
444   /*********************************************************************
445    * void XemsModifyReceiverConfigureNotify(event)
446    * Input:        event - Event Structure
447    * Output:       Print event info
448    * Algo:         Use event info to move/resize the window that
449    *               originated the event.
450    * Side Effects:         Move/resize the window that originated event.
451    *********************************************************************/
452   void XemsModifyReceiverConfigureNotify(event)
453   XEvent *event;
454   {
455     XMoveResizeWindow(CmInfo->cm_display, event->xconfigure.window,
456                       event->xconfigure.x, event->xconfigure.y,
457                       event->xconfigure.width, event->xconfigure.height);
458
459     XFlush(CmInfo->cm_display);
460   }
461
462
463   /*********************************************************************
464    * void XemsModifyReceiverMapNotify(event)
465    * Input:        event - Event Structure
466    * Output:       None
467    * Algo:         Use event info to map the window that originated event.
468    * Side Effects:         Map window.
469    *********************************************************************/
470   void XemsModifyReceiverMapNotify(event)
471   XEvent *event;
472   {
473     XMapWindow(CmInfo->cm_display, event->xmap.window);
474   }
475
476   /*********************************************************************
477    * void XemsModifyReceiverUnmapNotify(event)
478    * Input:        event - Event Structure
479    * Output:       None
480    * Algo:         Use event info to unmap window that originated event.
481    * Side Effects:         Unmap window.
482    *********************************************************************/
483   void XemsModifyReceiverUnmapNotify(event)
484   XEvent *event;
485   {
486     XUnmapWindow(CmInfo->cm_display, event->xunmap.window);
487   }
488
489
490   /*********************************************************************
491    * Window XemsFindWindow(dpy, top, name)  [Extracted from xsendevent,
492    *                                         Rod Whitby, Austek Microsystems]
493    * Input:        dpy - Ptr to display struct
494    *               top - Window id of toplevel widget
495    *               name - char string, containing name to search for
496    * Output:       Returns window id of widget whose name is 'name'
497    * Algo:         Routine to locate a window with a given name on a display.
498    *               If no window with the given name is found, 0 is returned.
499    *               If more than one window has the given name, the first
500    *               one found will be returned.  Only top and its subwindows
501    *               are looked at.  Normally, top should be the RootWindow.
502    *
503    * Side Effects:         None
504    *********************************************************************/
505   Window XemsFindWindow(dpy, top, name)
506   Display *dpy;
507   Window top;
508   char *name;
509   {
510     Window *children, dummy;       /* win ids */
511     unsigned int nchildren;        /* number of children */
512     int i;                         /* loop index */
```

```
513     Window w = 0;                   /* temp wind id */
514     char *window_name;              /* ptr to str, temporary */
515
516     /* Compare string names at this level */
517     if (XFetchName(dpy, top, &window_name) && !strcmp(window_name, name))
518       return(top); /* if they match, return win id */
519
520     /* Inquire the about widget children, request pointer and number of */
521     /* children */
522     if (!XQueryTree(dpy, top, &dummy, &dummy, &children, &nchildren))
523       return(0);
524
525     /* Do this recursively, for every child */
526     for (i=0; i<nchildren; i++) {
527       w = XemsFindWindow(dpy, children[i], name);
528       if (w)
529         break;
530     }
531     if (children) XFree ((char *)children); /* free children ids */
532     return(w);
533   }
534
535
536
537   /*********************************************************************
538    * void XemsSendEvent(e)
539    * Input:        e - event structure
540    * Output:       None
541    * Algo:         Sends an XEvent structure and its 2 associated trailer
542                    messages: time and name
543    * Side Effects:
544    *********************************************************************/
545   void XemsSendEvent(e)
546   XEvent *e;
547   {
548     XEvent *cme = XtNew(XEvent);
549     XEvent *cme2 = XtNew(XEvent);
550
551     XSendEvent(CmInfo->cm_display,
552                CmInfo->ed_window,True,ALLEVENTMASK,e);
553
554     /* Time trailer message */
555     XemsCreateTimeTrailer(cme2,
556                           CmInfo->cm_display,
557                           CmInfo->cm_window);
558
559     XSendEvent(CmInfo->cm_display,
560                CmInfo->ed_window,True,ALLEVENTMASK,cme2);
561
562     /* Name trailer message */
563     XemsCreateNameTrailer(cme,
564                           CmInfo->cm_display,
565                           CmInfo->cm_window,
566                           XtName(XtWindowToWidget(CmInfo->cm_display,
567                                                   e->xany.window)));
568     XSendEvent(CmInfo->cm_display,
569                CmInfo->ed_window,True,ALLEVENTMASK,cme);
570
571     XFlush(CmInfo->cm_display);
572     XFree(cme);
573     XFree(cme2);
574   }
575
576   /*********************************************************************
577    * void XemsCreateNameTrailer()
578    * Input:        ev - Event Struct (return)
579    *               disp - Ptr to display struct
580    *               win - Window id that originated the event
581    *               str - String to be stored as data
582    * Output:       None
583    * Algo:         Allocates ClientMessage event and stores name in it
584    * Side Effects:         Allocates new event struct
585    *********************************************************************/
586   void XemsCreateNameTrailer(ev,disp,win,str)
587   XEvent *ev;       /* Allocated event struct to be sent as message*/
588   Display *disp;    /* Display of this app */
589   Window win;       /* window that originated the event */
590   char *str;        /* string to be stored as data */
591   {
592     ev->type = ClientMessage;     /* Copy info from parameters */
593     ev->xclient.display = disp;
594     ev->xclient.window = win;
595     ev->xclient.message_type = XInternAtom(disp, "NONE", False);
596
597     /* If I changed Atom type the data gets screwed, better understand */
598     /* this atom stuff */
599     ev->xclient.format = 8;
600
601     strcpy(ev->xclient.data.b, str);      /* Copy name to data section */
602     ev->xclient.data.b[strlen(str)+1] = '\0';  /* zero terminate the string */
603   }
604
605   /*********************************************************************
606    * void XemsCreateTimeTrailer()
607    * Input:        ev - Event struct (return)
608    *               disp - Ptr to display struct
609    *               win - Window id that originated event
610    * Output:       None
611    * Algo:          Allocates ClientMessage event and stores system time
612    * Side Effects:         Allocates new event struct
613    *********************************************************************/
614   void XemsCreateTimeTrailer(ev,disp,win,str)
615   XEvent *ev;       /* Allocated event struct to be sent as message*/
616   Display *disp;    /* Display of this app */
617   Window win;       /* window that originated the event */
618   {
619     ev->type = ClientMessage;
620     ev->xclient.display = disp;
621     ev->xclient.window = win;
622     ev->xclient.message_type = XInternAtom(disp, "NONE", False);
623
624     /* If I changed Atom type the data gets screwed, better understand */
625     /* this atom stuff */
626     ev->xclient.format = 32;
627
628     ev->xclient.data.l[0] = GetCurrentTime(); /* Store time as long integer */
629     /* printf("Time %lu\n\n",  ev->xclient.data.l[0]); */
630   }
631
632
633
634   /*
635    * All these routines operate with global table WinTable
636    * These routines ARE NOT USED in this version
637    */
638
639   /*
640   void XemsInitWinTable()
```

```
641  {
642    WinTable.last_entry = 0;
643  }
644
645  void XemsInsertWinTable(win)
646  Window win;
647  {
648    if (WinTable.last_entry < MAX_WIN_TABLE_ENTRIES)
649      {
650        WinTable.array[WinTable.last_entry] = win;
651        WinTable.last_entry++;
652      }
653  }
654
655  void XemsPrintWinTable()
656  {
657    int i;
658
659    for(i=0;i<WinTable.last_entry;i++)
660      printf("[%d] %x\n",i,WinTable.array[i]);
661  }
662
663  Window XemsReturnEntryWinTable(entry)
664  int entry;
665  {
666    return (WinTable.array[entry]);
667  }
668
669  */
670
671
```

```
# use ANSI C Compiler
CC = /usr/lang/acc

NAME     = ed
BINARIES = ed.o util.o queue.o ui.o  file.o uiState.o
           uiAction.o uiMenu.o uiMessage.o XemsEvent.o uiFilter.o
#CFLAGS   = -I/natasha/sys2/X11R5/usr/include -I/cs/src/motif -I/natasha/sys2/X11R5/mit/lib/Xt

CFLAGS   =  -I/usr/include -I/cs/include/motif

DEFINES  = -DINCLUDE_ALLOCA_H -g
#LFLAGS   = -L/natasha/sys2/X11R5/usr/lib  -L/cs/lib/motif
LFLAGS   = -L/usr/lib -L/cs/lib/motif
LIB2     =  /usr/lang/SC1.0/libm.a -lXmu /u/spr/motif/libXm.a -lXt -lXext -lX11

#LIB2     = -lXmu -lXt -lXext -lX11 /cs/src/motif/lib/libXmdebug.a
#LIB2     = /cs/src/motif/lib/libXmdebug.a /natasha/sys2/X11R5/usr/lib/libXmu.a /u/ew/pro/libx/xtmain/libXt.a /natasha/sys2/X11R5/usr/lib/libXext.a /natasha/sys2/X11R5/usr/lib/libX11.a


# Define a rule for converting .c files into o. files
.c.o:      ; ${CC} -c $(CFLAGS) $(DEFINES) ${LFLAGS} $*.c


${NAME} : $(BINARIES)
        ${CC} -o ${NAME} ${CFLAGS} $(DEFINES) ${DEBUG} ${BINARIES} ${LFLAGS} ${LIB2}
```

```
1  /*****************************************************************************
2   * ed.h
3   *
4   *****************************************************************************/
5
6  /* Prototypes */
7
8  /* Modify Routines */
9  void EdAppMainLoop();      /* Special Main Loop for ED application */
10 void EdModifyEvent();      /* Check for events that are coming */
11 void EdModifySetup();      /* Setup initial state */
12
13 /********************/     /* To be deleted */
14 void EdModifySender();
15 /********************/
16
17 void EdModifyReceiver();  /* Filter events that are coming */
18 void EdModifyWindowId();  /* Add special code to win id */
19
20 /* Send Event Routines */
21 void EdSendEventQueue();  /* Send an event to CM */
22 void EdCreateMessage();   /* Create Client Message Event */
23 Bool EdSendMessage();     /* Send Client Message Event */
24
25 /* Other utility routines */
26 Window EdSelectWindow();     /* Grabs cursor, to select window to be tracked*/
27 void   EdCheckValidWindow(); /* Check if window id is valid */
28 void   BadWindow_Handler();  /* Exits program on bad window err */
29 int    CheckFilterEvent();   /* Checks if particular event has been filtered */
```

```
/*************************************************************************
 * ed.c
 *
 *************************************************************************/
#include <stdio.h>
#include <X11/cursorfont.h>
#include <X11/Intrinsic.h>

#include "XemsEvent.h"
#include "queue.h"
#include "ed.h"
#include "util.h"

/* Might put this into a struct for each client */
EventQueue Q;      /* Queue that stores events received */

GlobalInfo ed_info = {False,0,0,NORMAL};  /* Stucture that contains */
GlobalInfo *EdInfo = &ed_info;            /* disp/win state info */
Bool  print_flag = True;
extern long *filter_mask;

/*************************************************************************
 * void EdAppMainLoop()
 *
 * Input:        app - X Application Context
 * Output:       None
 * Algo:         Intercept event information between XtAppNextEvent()
 *               and XtDispatchEvent()
 * Side Effects:         Does not modify events produced locally (within
 *                       the application). Responds to messages from
 *                       the CM and stores events from it also.
 *************************************************************************/
void EdAppMainLoop(app)
XtAppContext app;
{
    XEvent event;

    for (;;) {
        XtAppNextEvent(app, &event);    /* Take event from Event Queue */
        EdModifyEvent(&event);          /* Intercept event */
        XtDispatchEvent(&event);        /* Dispatch events to widgets */
    }
}

/*************************************************************************
 * void EdModifyEvent()
 *
 * Input:        event - pointer to event structure
 * Output:       None
 * Algo:         Setup information (window, display) and check for
 *               either messages or events from the CM.
 * Side Effects:         Responds to events and messages from CM.
 *                       EdModifyReceiver() handles this.
 *************************************************************************/
void EdModifyEvent(event)
XEvent *event;
{
  EdModifySetup(event);
  /* EdModifySender(event); */
  EdModifyReceiver(event);
}

/*************************************************************************
 * void EdModifySetup()
 *
 * Input:        event - pointer to event structure
 * Output:       None
 * Algo:         Initialize flag
 * Side Effects:         Modifies EdInfo global structure.
 *************************************************************************/
void EdModifySetup(event)
XEvent *event;
{
   * Initialize information */
  if (EdInfo->init == False)
    {
      EdInfo->init = True;
    }

}

/*************************************************************************
 * void EdModifySender(event)
 * Input:        event -
 * Output:
 * Algo:
 * Side Effects:
 *************************************************************************/
void EdModifySender(event)
XEvent *event;
{

}

static int client_message_state = -1; /* Setup message state */
/*************************************************************************
 * void EdModifyReceiver()
 *
 * Input:        event - pointer to event struct to be processed
 * Output:
 * Algo:
 * Side Effects:
 *************************************************************************/
void EdModifyReceiver(event)
XEvent *event;
{

  /* If event is xsent and is not originated by the Event Daemon */
  if ((event->xany.send_event) &&
      ((event->xany.window & HIGH_MASK) !=
       (EdInfo->ed_window & HIGH_MASK)))
    {
      if (event->type != ClientMessage)   /* If it's an event queue it */
        {
          /* copy high order bits of ED's window to the event */
          /* so that it's unique id'd as coming from ED */
          if (CheckFilterEvent(event->type))
            {

              if (print_flag)
                {
                  PrintEvent(event);
                  printf("----------------\n");
                  printf("Received Event from Application\n");
                }

              EdModifyWindowId(event);
              EnQueue(&Q,CreateNode(event));
```

```
129                client_message_state = 0;
130              }
131          }
132        else
133          {
134            client_message_state++;
135            /* count states */
136            if (client_message_state == 1)
137              {
138                AddTime(&Q,event->xclient.data.l[0]);
139                /* printf("Time %lu\n", event->xclient.data.l[0]); */
140                 * printf("Adding Time\n"); */
141              }
142            else
143              {
144                AddName(&Q,event->xclient.data.b);
145                /* printf("Adding Name\n");*/
146                client_message_state = 0;
147              }
148          }
149      }
150
151  }
152
153  /*********************************************************************
154   * void EdModifyWindowId()
155   * Input:
156   * Output:
157   * Algo:
158   * Side Effects:
159   *********************************************************************/
160  void EdModifyWindowId(event)
161  XEvent *event;
162  {
163    /* These are events that have the field "window" somewhere else */
164    switch(event->type)
165      {
166      case CirculateNotify:
167        bcopy(&(EdInfo->ed_window), &(event->xcirculate.window),2);
168        event->xcirculate.window |= MIDDLE_MASK;
169        break;
170      case CirculateRequest:
171        bcopy(&(EdInfo->ed_window), &(event->xcirculaterequest.window),2);
172        event->xcirculaterequest.window |= MIDDLE_MASK;
173        break;
174      case ConfigureNotify:
175        bcopy(&(EdInfo->ed_window), &(event->xconfigure.window),2);
176        event->xconfigure.window |= MIDDLE_MASK;
177        break;
178      case ConfigureRequest:
179        bcopy(&(EdInfo->ed_window), &(event->xconfigurerequest.window),2);
180        event->xconfigurerequest.window |= MIDDLE_MASK;
181        break;
182      case CreateNotify:
183        bcopy(&(EdInfo->ed_window), &(event->xcreatewindow.window),2);
184        event->xcreatewindow.window |= MIDDLE_MASK;
185        break;
186      case DestroyNotify:
187        bcopy(&(EdInfo->ed_window), &(event->xdestroywindow.window),2);
188        event->xdestroywindow.window |= MIDDLE_MASK;
189        break;
190      case GravityNotify:
191        bcopy(&(EdInfo->ed_window), &(event->xgravity.window),2);
192        event->xgravity.window |= MIDDLE_MASK;
193        break;
194      case MapNotify:
195        bcopy(&(EdInfo->ed_window), &(event->xmap.window),2);
196        event->xmap.window |= MIDDLE_MASK;
197        break;
198      case UnmapNotify:
199        bcopy(&(EdInfo->ed_window), &(event->xunmap.window),2);
200        event->xunmap.window |= MIDDLE_MASK;
201        break;
202      case MapRequest:
203        bcopy(&(EdInfo->ed_window), &(event->xmaprequest.window),2);
204        event->xmaprequest.window |= MIDDLE_MASK;
205        break;
206      case ReparentNotify:
207        bcopy(&(EdInfo->ed_window), &(event->xreparent.window),2);
208        event->xreparent.window |= MIDDLE_MASK;
209        break;
210      default:
211        bcopy(&(EdInfo->ed_window), &(event->xany.window),2);
212        event->xany.window |= MIDDLE_MASK;
213      }
214  }
215
216  /*********************************************************************
217   * void EdSendEventQueue()
218   * Input:
219   * Output:
220   * Algo:
221   * Side Effects:
222   *********************************************************************/
223  void EdSendEventQueue(p)
224  QEvent *p;
225  {
226    Display *disp;
227    Window window;
228
229    if (CheckFilterEvent(p->event->type)) /* True: not filtered out */
230      {
231        disp = p->event->xany.display;    /* ED's display */
232        window = p->event->xany.window;   /* destination window */
233        XSendEvent(disp,EdInfo->cm_window,True,ALLEVENTMASK,p->event);
234        XFlush(disp);
235      }
236  }
237
238
239  /*********************************************************************
240   * void EdCreateMessage()
241   * Allocates ClientMessage event and stores info in it
242   * Used for sending change of state messages.
243   * Input:
244   * Output:
245   * Algo:
246   * Side Effects:
247   *********************************************************************/
248  void EdCreateMessage(ev,disp,win,st)
249  XEvent *ev;
250  Display *disp;
251  Window win;
252  int st;
253  {
254    ev->type = ClientMessage;
255    ev->xclient.display = disp;
256    ev->xclient.window = win;
```

```
257     ev->xclient.message_type = XInternAtom(disp, "NONE", False);
258     /* If I changed Atom type the data gets screwed, better understand *
259     /* this atom stuff */
260     ev->xclient.format = 8;
261     /*bzero(ev->xclient.data,20);*/
262     /*strncpy(ev->xclient.data.b, "s:0",2);*/
263     EdModifyWindowId(ev);
264
265     sprintf(ev->xclient.data.b,"s%c", st);
266     bcopy(&(EdInfo->ed_window),&(ev->xclient.data.b[2]),4);
267   }
268
269   /*************************************************************************
270    * void EdSendMessage()
271    * Input:
272    * Output:
273    * Algo:
274    * Side Effects:
275    *
276    *************************************************************************/
277   Bool EdSendMessage(disp,win,st)
278   Display *disp;
279   Window win;
280   int st;
281   {
282     XEvent *msg;
283     msg = XtNew(XEvent);
284
285     if (win == NULL)
286       {
287         MessagePrint("Select Window first\n");
288         return False;
289       }
290     else
291       {
292         EdCheckValidWindow(win);
293
294         EdCreateMessage(msg,disp,win,st);
295         XSendEvent(disp,win,True,ALLEVENTMASK,msg);
296
297         XFlush(disp);
298         return True;
299       }
300   }
301
302   /*************************************************************************
303    * Window EdSelectWindow(disp)
304    * Input:
305    * Output:
306    * Algo:
307    * Side Effects:
308    *
309    *************************************************************************/
310   Window EdSelectWindow(disp)
311   Display *disp;
312   {
313     int screen = DefaultScreen(EdInfo->ed_display);
314     int status;
315     Cursor cursor;
316     XEvent event;
317     Window target_win = None, root = RootWindow(disp,screen);
318     int buttons = 0;
319
320     /* Make the target cursor */
321     cursor = XCreateFontCursor(disp, XC_crosshair);
322
323     /* Grab the pointer using target cursor, letting it room all over */
324     status = XGrabPointer(disp, root, False,
325                           ButtonPressMask|ButtonReleaseMask, GrabModeSync,
326                           GrabModeAsync, root, cursor, CurrentTime);
327     if (status != GrabSuccess)
328       MessagePrint("Can't grab the mouse.\n");
329
330     /* Let the user select a window... */
331     while ((target_win == None) || (buttons != 0))
332       {
333         /* allow one more event */
334         XAllowEvents(disp, SyncPointer, CurrentTime);
335         XWindowEvent(disp, root, ButtonPressMask|ButtonReleaseMask, &event);
336         switch (event.type)
337           {
338           case ButtonPress:
339             if (target_win == None)
340               {
341                 target_win = event.xbutton.subwindow; /* window selected */
342                 if (target_win == None) target_win = root;
343               }
344             buttons++;
345             break;
346           case ButtonRelease:
347             if (buttons > 0) /*there may have been some down before we started */
348               buttons--;
349             break;
350           }
351       }
352     XUngrabPointer(disp, CurrentTime);       /* Done with pointer */
353     return(target_win);
354   }
355
356   /*************************************************************************
357    * void EdCheckValidWindow()
358    * Input:
359    * Output:
360    * Algo:
361    * Side Effects:
362    *
363    *************************************************************************/
364   void EdCheckValidWindow(win)
365   Window win;
366   {
367     Window root;
368     int x, y;
369     unsigned width, height, bw, depth;
370     int (*old_handler)();
371
372     old_handler = XSetErrorHandler(BadWindow_Handler);
373     XGetGeometry(EdInfo->ed_display, win,
374                  &root, &x, &y, &width, &height, &bw, &depth);
375     XSync(EdInfo->ed_display, False);
376     XSetErrorHandler(old_handler);
377   }
378
379   /*************************************************************************
380    * void EdCheckValidWindow()
381    * Input:
382    * Output:
383    * Algo:
384    * Side Effects:
```

```
 *
 ************************************************************************
void BadWindow_Handler(disp, err)
Display *disp;
XErrorEvent *err;
{
  printf("No such window id 0x%lx\n",err->resourceid);
  exit(1);
}

 ************************************************************************
 * void EdCheckValidWindow(win)
 * Returns False if it has been popdown (toggle button)
 * Return  True if it's up.
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 ************************************************************************
int CheckFilterEvent(type)
int type;
{
  if (type < 31)
    {
      /*
        printf("Mask %x\n", filter_mask[0]);
        printf("State %d \n", (1L << type) & filter_mask[0]);
        */
      return !((1L << type) & filter_mask[0]);
    }
  else
    {
      /*
        printf("Mask %x\n", filter_mask[1]);
        printf("State %d \n", (1L << type) & filter_mask[1]);
        */
      return !((1L << type) & filter_mask[1]);
    }
}

```

```
1  /*************************************************************************
2   * file.h
3   *
4   *************************************************************************
5
6  void EdQueueToFile();   /* Converts Queue (in memory) to file */
7  void EdEventToFile();   /* Converts Event struct to file */
8  void EdFileToQueue();   /* Reads in file and converts it to queue */
9
10
11 /* These routines store each Event type to file format */
12 void StoreKeyPress();
13 void StoreKeyRelease();
14 void StoreButtonPress();
15 void StoreButtonRelease();
16 void StoreMotionNotify();
17 void StoreEnterNotify();
18 void StoreLeaveNotify();
19 void StoreFocusIn();
20 void StoreFocusOut();
21 void StoreKeymapNotify();
22 void StoreExpose();
23 void StoreGraphicsExpose();
24 void StoreNoExpose();
25 void StoreVisibilityNotify();
26 void StoreCreateNotify();
27 void StoreDestroyNotify();
28 void StoreUnmapNotify();
29 void StoreMapNotify();
30 void StoreMapRequest();
31 void StoreReparentNotify();
32 void StoreConfigureNotify();
33 void StoreConfigureRequest();
34 void StoreGravityNotify();
35 void StoreResizeRequest();
36 void StoreCirculateNotify();
37 void StoreCirculateRequest();
38 void StorePropertyNotify();
39 void StoreSelectionClear();
40 void StoreSelectionReques();
41 void StoreSelectionNotify();
42 void StoreColormapNotify();
43 void StoreClientMessage();
44 void StoreMappingNotify();
45
46
47
48 /* These routines convert events stored in file to queue (memory) */
49 void ReadKeyPress();
50 void ReadKeyRelease();
51 void ReadButtonPress();
52 void ReadButtonRelease();
53 void ReadMotionNotify();
54 void ReadEnterNotify();
55 void ReadLeaveNotify();
56 void ReadFocusIn();
57 void ReadFocusOut();
58 void ReadKeymapNotify();
59 void ReadExpose();
60 void ReadGraphicsExpose();
61 void ReadNoExpose();
62 void ReadVisibilityNotify();
63 void ReadCreateNotify();
64 void ReadDestroyNotify();
65 void ReadUnmapNotify();
66 void ReadMapNotify();
67 void ReadMapRequest();
68 void ReadReparentNotify();
69 void ReadConfigureNotify();
70 void ReadConfigureRequest();
71 void ReadGravityNotify();
72 void ReadResizeRequest();
73 void ReadCirculateNotify();
74 void ReadCirculateRequest();
75 void ReadPropertyNotify();
76 void ReadSelectionClear();
77 void ReadSelectionReques();
78 void ReadSelectionNotify();
79 void ReadColormapNotify();
80 void ReadClientMessage();
81 void ReadMappingNotify();
```

```
1   /*************************************************************************
2    * file.c
3    *
4    *************************************************************************/
5
6   #include <stdio.h>
7
8   #include "queue.h"
9   #include "ed.h"
10  #include "XemsEvent.h"
11  #include "file.h"
12
13  extern GlobalInfo *EdInfo;
14
15  /*************************************************************************
16   * void EdQueueToFile()
17   * Input:        q - event queue
18   * Output:       file "output.xes"
19   * Algo:         Traverses the queue and converts every event to file
20   * Side Effects:         Overwites contents of output.xes
21   *************************************************************************/
22  void EdQueueToFile(q)
23  EventQueue *q;
24  {
25    QEvent *p;
26    FILE *file;
27
28    file = fopen("output.xes","w");
29
30    p = q->front;
31    while (p!=NULL)
32      {
33        EdEventToFile(file, p);
34        p = p->next;
35      }
36    fclose(file);
37  }
38
39  /*************************************************************************
40   * void EdEventToFile()
41   * Input:        file - file stream to write file out
42   *               qevent - event to convert to file
43   * Output:       Event converted to file (output.xes)
44   * Algo:         Writes an event to file format
45   * Side Effects:         Overwrites on the file stream that is passed
46   *************************************************************************/
47  void EdEventToFile(file, qevent)
48  FILE *file;
49  QEvent *qevent;
50  {
51    switch(qevent->event->type)
52      {
53      case KeyPress:
54        StoreKeyPress(file,qevent);
55        break;
56      case KeyRelease:
57        StoreKeyRelease(file,qevent);
58        break;
59      case ButtonPress:
60        StoreButtonPress(file,qevent);
61        break;
62      case ButtonRelease:
63        StoreButtonRelease(file,qevent);
64        break;
65      case MotionNotify:
66        StoreMotionNotify(file,qevent);
67        break;
68      case EnterNotify:
69        StoreEnterNotify(file,qevent);
70        break;
71      case LeaveNotify:
72        StoreLeaveNotify(file,qevent);
73        break;
74      case FocusIn:
75        StoreFocusIn(file,qevent);
76        break;
77      case FocusOut:
78        StoreFocusOut(file,qevent);
79        break;
80      case KeymapNotify:
81        StoreKeymapNotify(file,qevent);
82        break;
83      case Expose:
84        StoreExpose(file,qevent);
85        break;
86      case GraphicsExpose:
87        StoreGraphicsExpose(file,qevent);
88        break;
89      case NoExpose:
90        StoreNoExpose(file,qevent);
91        break;
92      case VisibilityNotify:
93        StoreVisibilityNotify(file,qevent);
94        break;
95      case CreateNotify:
96        StoreCreateNotify(file,qevent);
97        break;
98      case DestroyNotify:
99        StoreDestroyNotify(file,qevent);
100       break;
101     case UnmapNotify:
102       StoreUnmapNotify(file,qevent);
103       break;
104     case MapNotify:
105       StoreMapNotify(file,qevent);
106       break;
107     case MapRequest:
108       StoreMapRequest(file,qevent);
109       break;
110     case ReparentNotify:
111       StoreReparentNotify(file,qevent);
112       break;
113     case ConfigureNotify:
114       StoreConfigureNotify(file,qevent);
115       break;
116     case ConfigureRequest:
117       StoreConfigureRequest(file,qevent);
118       break;
119     case GravityNotify:
120       StoreGravityNotify(file,qevent);
121       break;
122     case ResizeRequest:
123       StoreResizeRequest(file,qevent);
124       break;
125     case CirculateNotify:
126       StoreCirculateNotify(file,qevent);
127       break;
128     case CirculateRequest:
```

```
      StoreCirculateRequest(file,qevent);
      break;
    case PropertyNotify:
      StorePropertyNotify(file,qevent);
      break;
    case SelectionClear:
      StoreSelectionClear(file,qevent);
      break;
    case SelectionNotify:
      StoreSelectionNotify(file,qevent);
      break;
    case ColormapNotify:
      StoreColormapNotify(file,qevent);
      break;
    case ClientMessage:
      StoreClientMessage(file,qevent);
      break;
    case MappingNotify:
      StoreMappingNotify(file,qevent);
      break;
    }
}

/*************************************************************************
 * void EdFileToQueue(q)
 * Input:        file: output.xes
 * Output:       Event queue in memory
 * Algo:         Reads the file output.xes and creates a new queue from it
 * Side Effects:         Overwrites the current queue in memory
 *************************************************************************/
void EdFileToQueue(q)
EventQueue *q;
{
  int type;
  QEvent *p;
  FILE *file;
  QEvent *qevent;

  file = fopen("output.xes","r");

  while (fscanf(file,"%d",&type) != EOF)
    {
      qevent = XtNew(QEvent);
      qevent->event = XtNew(XEvent);
      qevent->name = XtMalloc(25);

      qevent->event->type = type;
      switch(type)
        {
        case KeyPress:
          ReadKeyPress(file,qevent);
          break;
        case KeyRelease:
          ReadKeyRelease(file,qevent);
          break;
        case ButtonPress:
          ReadButtonPress(file,qevent);
          break;
        case ButtonRelease:
          ReadButtonRelease(file,qevent);
          break;
        case MotionNotify:
          ReadMotionNotify(file,qevent);
          break;
        case EnterNotify:
          ReadEnterNotify(file,qevent);
          break;
        case LeaveNotify:
          ReadLeaveNotify(file,qevent);
          break;
        case FocusIn:
          ReadFocusIn(file,qevent);
          break;
        case FocusOut:
          ReadFocusOut(file,qevent);
          break;
        case KeymapNotify:
          ReadKeymapNotify(file,qevent);
          break;
        case Expose:
          ReadExpose(file,qevent);
          break;
        case GraphicsExpose:
          ReadGraphicsExpose(file,qevent);
          break;
        case NoExpose:
          ReadNoExpose(file,qevent);
          break;
        case VisibilityNotify:
          ReadVisibilityNotify(file,qevent);
          break;
        case CreateNotify:
          ReadCreateNotify(file,qevent);
          break;
        case DestroyNotify:
          ReadDestroyNotify(file,qevent);
          break;
        case UnmapNotify:
          ReadUnmapNotify(file,qevent);
          break;
        case MapNotify:
          ReadMapNotify(file,qevent);
          break;
        case MapRequest:
          ReadMapRequest(file,qevent);
          break;
        case ReparentNotify:
          ReadReparentNotify(file,qevent);
          break;
        case ConfigureNotify:
          ReadConfigureNotify(file,qevent);
          break;
        case ConfigureRequest:
          ReadConfigureRequest(file,qevent);
          break;
        case GravityNotify:
          ReadGravityNotify(file,qevent);
          break;
        case ResizeRequest:
          ReadResizeRequest(file,qevent);
          break;
        case CirculateNotify:
          ReadCirculateNotify(file,qevent);
          break;
        case CirculateRequest:
          ReadCirculateRequest(file,qevent);
          break;
        case PropertyNotify:
```

```
257             ReadPropertyNotify(file,qevent);
258             break;
259           case SelectionClear:
260             ReadSelectionClear(file,qevent);
261             break;
262           case SelectionNotify:
263             ReadSelectionNotify(file,qevent);
264             break;
265           case ColormapNotify:
266             ReadColormapNotify(file,qevent);
267             break;
268           case ClientMessage:
269             ReadClientMessage(file,qevent);
270             break;
271           case MappingNotify:
272             ReadMappingNotify(file,qevent);
273             break;
274           }
275         qevent->event->xany.send_event = True;
276         qevent->event->xany.display = EdInfo->ed_display;
277         EnQueue(q, qevent);
278       }
279 }
280
281
282 /*************************************************************************
283  * Store Routines (event structure to file)
284  *************************************************************************/
285 /*************************************************************************
286  * void StoreKeyPress()
287  *************************************************************************/
288 void StoreKeyPress(file, qevent)
289 FILE *file;
290 QEvent *qevent;
291 {
292   fprintf(file,"%d\n",qevent->event->xkey.type);
293   fprintf(file,"%s\n",qevent->name);
294   fprintf(file,"%lu\n",qevent->time);
295   fprintf(file,"%x\n",qevent->event->xkey.window);
296   fprintf(file,"%x\n",qevent->event->xkey.root);
297   fprintf(file,"%x\n",qevent->event->xkey.subwindow);
298   fprintf(file,"%d\n",qevent->event->xkey.time);
299   fprintf(file,"%d\n",qevent->event->xkey.x);
300   fprintf(file,"%d\n",qevent->event->xkey.y);
301   fprintf(file,"%d\n",qevent->event->xkey.x_root);
302   fprintf(file,"%d\n",qevent->event->xkey.y_root);
303   fprintf(file,"%d\n",qevent->event->xkey.state);
304   fprintf(file,"%d\n",qevent->event->xkey.keycode);
305   fprintf(file,"%d\n\n",qevent->event->xkey.same_screen);
306 }
307
308 /*************************************************************************
309  * void StoreKeyRelease()
310  *************************************************************************/
311 void StoreKeyRelease(file, qevent)
312 FILE *file;
313 QEvent *qevent;
314 {
315   /* It has the same info */
316   StoreKeyPress(file,qevent);
317 }
318
319 /*************************************************************************
320  * void StoreButtonPress()
321  *************************************************************************/
322 void StoreButtonPress(file, qevent)
323 FILE *file;
324 QEvent *qevent;
325 {
326   fprintf(file,"%d\n",qevent->event->xbutton.type);
327   fprintf(file,"%s\n",qevent->name);
328   fprintf(file,"%lu\n",qevent->time);
329   fprintf(file,"%x\n",qevent->event->xbutton.window);
330   fprintf(file,"%x\n",qevent->event->xbutton.root);
331   fprintf(file,"%x\n",qevent->event->xbutton.subwindow);
332   fprintf(file,"%d\n",qevent->event->xbutton.time);
333   fprintf(file,"%d\n",qevent->event->xbutton.x);
334   fprintf(file,"%d\n",qevent->event->xbutton.y);
335   fprintf(file,"%d\n",qevent->event->xbutton.x_root);
336   fprintf(file,"%d\n",qevent->event->xbutton.y_root);
337   fprintf(file,"%d\n",qevent->event->xbutton.state);
338   fprintf(file,"%d\n",qevent->event->xbutton.button);
339   fprintf(file,"%d\n\n",qevent->event->xbutton.same_screen);
340 }
341
342
343 /*************************************************************************
344  * void StoreButtonRelease()
345  *************************************************************************/
346 void StoreButtonRelease(file, qevent)
347 FILE *file;
348 QEvent *qevent;
349 {
350   /* It has the same info */
351   StoreButtonPress(file,qevent);
352 }
353
354 /*************************************************************************
355  * void StoreMotionNotify()
356  *************************************************************************/
357 void StoreMotionNotify(file, qevent)
358 FILE *file;
359 QEvent *qevent;
360 {
361   fprintf(file,"%d\n",qevent->event->xmotion.type);
362   fprintf(file,"%s\n",qevent->name);
363   fprintf(file,"%lu\n",qevent->time);
364   fprintf(file,"%x\n",qevent->event->xmotion.window);
365   fprintf(file,"%x\n",qevent->event->xmotion.root);
366   fprintf(file,"%x\n",qevent->event->xmotion.subwindow);
367   fprintf(file,"%d\n",qevent->event->xmotion.time);
368   fprintf(file,"%d\n",qevent->event->xmotion.x);
369   fprintf(file,"%d\n",qevent->event->xmotion.y);
370   fprintf(file,"%d\n",qevent->event->xmotion.x_root);
371   fprintf(file,"%d\n",qevent->event->xmotion.y_root);
372   fprintf(file,"%d\n",qevent->event->xmotion.state);
373   fprintf(file,"%c\n",qevent->event->xmotion.is_hint);
374   fprintf(file,"%d\n\n",qevent->event->xmotion.same_screen);
375 }
376
377 /*************************************************************************
378  * void StoreEnterNotify()
379  *************************************************************************/
380 void StoreEnterNotify(file, qevent)
381 FILE *file;
382 QEvent *qevent;
383 {
384   fprintf(file,"%d\n",qevent->event->xcrossing.type);
```

```
  fprintf(file,"%s\n",qevent->name);
  fprintf(file,"%lu\n",qevent->time);
  fprintf(file,"%x\n",qevent->event->xcrossing.window);
  fprintf(file,"%x\n",qevent->event->xcrossing.root);
  fprintf(file,"%x\n",qevent->event->xcrossing.subwindow);
  fprintf(file,"%d\n",qevent->event->xcrossing.time);
  fprintf(file,"%d\n",qevent->event->xcrossing.x);
  fprintf(file,"%d\n",qevent->event->xcrossing.y);
  fprintf(file,"%d\n",qevent->event->xcrossing.x_root);
  fprintf(file,"%d\n",qevent->event->xcrossing.y_root);
  fprintf(file,"%d\n",qevent->event->xcrossing.mode);
  fprintf(file,"%d\n",qevent->event->xcrossing.detail);
  fprintf(file,"%d\n",qevent->event->xcrossing.same_screen);
  fprintf(file,"%d\n",qevent->event->xcrossing.focus);
  fprintf(file,"%d\n\n",qevent->event->xcrossing.state);
}

/*************************************************************************
 * void StoreLeaveNotify()
 *************************************************************************/
void StoreLeaveNotify(file, qevent)
FILE *file;
QEvent *qevent;
{
  /* It has the same info */
  StoreEnterNotify(file,qevent);
}


/*************************************************************************
 * void StoreFocusIn()
 *************************************************************************/
void StoreFocusIn(file, qevent)
FILE *file;
QEvent *qevent;
{
  fprintf(file,"%d\n",qevent->event->xfocus.type);
  fprintf(file,"%s\n",qevent->name);
  fprintf(file,"%lu\n",qevent->time);
  fprintf(file,"%x\n",qevent->event->xfocus.window);
  fprintf(file,"%d\n",qevent->event->xfocus.mode);
  fprintf(file,"%d\n\n",qevent->event->xfocus.detail);
}


/*************************************************************************
 * void StoreFocusOut()
 *************************************************************************/
void StoreFocusOut(file, qevent)
FILE *file;
QEvent *qevent;
{
  /* It has the same info */
  StoreFocusIn(file,qevent);
}


/*************************************************************************
 * void StoreKeymapNotify()
 *************************************************************************/
void StoreKeymapNotify(file, qevent)
FILE *file;
QEvent *qevent;
{
  int i;

  fprintf(file,"%d\n",qevent->event->xkeymap.type);
  fprintf(file,"%s\n",qevent->name);
  fprintf(file,"%lu\n",qevent->time);
  fprintf(file,"%x\n",qevent->event->xkeymap.window);

  for (i=0; i<32; i++)
    fprintf(file,"%x\n",qevent->event->xkeymap.key_vector[i]);

  fprintf(file,"\n");
}


/*************************************************************************
 * void StoreExpose()
 *************************************************************************/
void StoreExpose(file, qevent)
FILE *file;
QEvent *qevent;
{
  fprintf(file,"%d\n",qevent->event->xexpose.type);
  fprintf(file,"%s\n",qevent->name);
  fprintf(file,"%lu\n",qevent->time);
  fprintf(file,"%x\n",qevent->event->xexpose.window);
  fprintf(file,"%d\n",qevent->event->xexpose.x);
  fprintf(file,"%d\n",qevent->event->xexpose.y);
  fprintf(file,"%d\n",qevent->event->xexpose.width);
  fprintf(file,"%d\n",qevent->event->xexpose.height);
  fprintf(file,"%d\n\n",qevent->event->xexpose.count);
}


/*************************************************************************
 * void StoreGraphicsExpose()
 *************************************************************************/
void StoreGraphicsExpose(file, qevent)
FILE *file;
QEvent *qevent;
{
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.type);
  fprintf(file,"%s\n",qevent->name);
  fprintf(file,"%lu\n",qevent->time);
  fprintf(file,"%x\n",qevent->event->xgraphicsexpose.drawable);
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.x);
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.y);
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.width);
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.height);
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.count);
  fprintf(file,"%d\n",qevent->event->xgraphicsexpose.major_code);
  fprintf(file,"%d\n\n",qevent->event->xgraphicsexpose.minor_code);
}


/*************************************************************************
 * void StoreNoExpose()
 *************************************************************************/
void StoreNoExpose(file, qevent)
FILE *file;
QEvent *qevent;
{
  fprintf(file,"%d\n",qevent->event->xnoexpose.type);
  fprintf(file,"%s\n",qevent->name);
  fprintf(file,"%lu\n",qevent->time);
```

```
513     fprintf(file,"%x\n",qevent->event->xnoexpose.drawable);
514     fprintf(file,"%d\n",qevent->event->xnoexpose.major_code);
515     fprintf(file,"%d\n\n",qevent->event->xnoexpose.minor_code);
516   }
517
518
519   /*************************************************************************
520    * void StoreVisibilityNotify()
521    *************************************************************************/
522   void StoreVisibilityNotify(file, qevent)
523   FILE *file;
524   QEvent *qevent;
525   {
526     fprintf(file,"%d\n",qevent->event->xvisibility.type);
527     fprintf(file,"%s\n",qevent->name);
528     fprintf(file,"%lu\n",qevent->time);
529     fprintf(file,"%x\n",qevent->event->xvisibility.window);
530     fprintf(file,"%d\n\n",qevent->event->xvisibility.state);
531   }
532
533
534   /*************************************************************************
535    * void StoreCreateNotify()
536    *************************************************************************/
537   void StoreCreateNotify(file, qevent)
538   FILE *file;
539   QEvent *qevent;
540   {
541     fprintf(file,"%d\n",qevent->event->xcreatewindow.type);
542     fprintf(file,"%s\n",qevent->name);
543     fprintf(file,"%lu\n",qevent->time);
544     fprintf(file,"%x\n",qevent->event->xcreatewindow.parent);
545     fprintf(file,"%x\n",qevent->event->xcreatewindow.window);
546     fprintf(file,"%d\n",qevent->event->xcreatewindow.x);
547     fprintf(file,"%d\n",qevent->event->xcreatewindow.y);
548     fprintf(file,"%d\n",qevent->event->xcreatewindow.width);
549     fprintf(file,"%d\n",qevent->event->xcreatewindow.height);
550     fprintf(file,"%d\n\n",qevent->event->xcreatewindow.override_redirect);
551   }
552
553
554   /*************************************************************************
555    * void StoreDestroyNotify()
556    *************************************************************************/
557   void StoreDestroyNotify(file, qevent)
558   FILE *file;
559   QEvent *qevent;
560   {
561     fprintf(file,"%d\n",qevent->event->xdestroywindow.type);
562     fprintf(file,"%s\n",qevent->name);
563     fprintf(file,"%lu\n",qevent->time);
564     fprintf(file,"%x\n",qevent->event->xdestroywindow.event);
565     fprintf(file,"%x\n\n",qevent->event->xdestroywindow.window);
566   }
567
568
569   /*************************************************************************
570    * void StoreUnmapNotify()
571    *************************************************************************/
572   void StoreUnmapNotify(file, qevent)
573   FILE *file;
574   QEvent *qevent;
575   {
576     fprintf(file,"%d\n",qevent->event->xunmap.type);
577     fprintf(file,"%s\n",qevent->name);
578     fprintf(file,"%lu\n",qevent->time);
579     fprintf(file,"%x\n",qevent->event->xunmap.event);
580     fprintf(file,"%x\n",qevent->event->xunmap.window);
581     fprintf(file,"%d\n\n",qevent->event->xunmap.from_configure);
582   }
583
584
585   /*************************************************************************
586    * void StoreMapNotify()
587    *************************************************************************/
588   void StoreMapNotify(file, qevent)
589   FILE *file;
590   QEvent *qevent;
591   {
592     fprintf(file,"%d\n",qevent->event->xmap.type);
593     fprintf(file,"%s\n",qevent->name);
594     fprintf(file,"%lu\n",qevent->time);
595     fprintf(file,"%x\n",qevent->event->xmap.event);
596     fprintf(file,"%x\n",qevent->event->xmap.window);
597     fprintf(file,"%d\n\n",qevent->event->xmap.override_redirect);
598   }
599
600   /*************************************************************************
601    * void StoreMapRequest()
602    *************************************************************************/
603   void StoreMapRequest(file, qevent)
604   FILE *file;
605   QEvent *qevent;
606   {
607     fprintf(file,"%d\n",qevent->event->xmaprequest.type);
608     fprintf(file,"%s\n",qevent->name);
609     fprintf(file,"%lu\n",qevent->time);
610     fprintf(file,"%x\n",qevent->event->xmaprequest.parent);
611     fprintf(file,"%x\n\n",qevent->event->xmaprequest.window);
612   }
613
614   /*************************************************************************
615    * void StoreReparentNotify()
616    *************************************************************************/
617   void StoreReparentNotify(file, qevent)
618   FILE *file;
619   QEvent *qevent;
620   {
621     fprintf(file,"%d\n",qevent->event->xreparent.type);
622     fprintf(file,"%s\n",qevent->name);
623     fprintf(file,"%lu\n",qevent->time);
624     fprintf(file,"%x\n",qevent->event->xreparent.event);
625     fprintf(file,"%x\n",qevent->event->xreparent.window);
626     fprintf(file,"%x\n",qevent->event->xreparent.parent);
627     fprintf(file,"%d\n",qevent->event->xreparent.x);
628     fprintf(file,"%d\n",qevent->event->xreparent.y);
629     fprintf(file,"%d\n\n",qevent->event->xreparent.override_redirect);
630   }
631
632   /*************************************************************************
633    * void StoreConfigureNotify()
634    *************************************************************************/
635   void StoreConfigureNotify(file, qevent)
636   FILE *file;
637   QEvent *qevent;
638   {
639     fprintf(file,"%d\n",qevent->event->xconfigure.type);
640     fprintf(file,"%s\n",qevent->name);
```

```
641     fprintf(file,"%lu\n",qevent->time);
642     fprintf(file,"%x\n",qevent->event->xconfigure.event);
643     fprintf(file,"%x\n",qevent->event->xconfigure.window);
644     fprintf(file,"%d\n",qevent->event->xconfigure.x);
645     fprintf(file,"%d\n",qevent->event->xconfigure.y);
646     fprintf(file,"%d\n",qevent->event->xconfigure.width);
647     fprintf(file,"%d\n",qevent->event->xconfigure.height);
648     fprintf(file,"%d\n",qevent->event->xconfigure.border_width);
649     fprintf(file,"%x\n",qevent->event->xconfigure.above);
650     fprintf(file,"%d\n\n",qevent->event->xconfigure.override_redirect);
651   }
652
653   /*************************************************************************
654    * void StoreConfigureRequest()
655    *************************************************************************/
656   void StoreConfigureRequest(file, qevent)
657   FILE *file;
658   QEvent *qevent;
659   {
660     fprintf(file,"%d\n",qevent->event->xconfigurerequest.type);
661     fprintf(file,"%s\n",qevent->name);
662     fprintf(file,"%lu\n",qevent->time);
663     fprintf(file,"%x\n",qevent->event->xconfigurerequest.parent);
664     fprintf(file,"%x\n",qevent->event->xconfigurerequest.window);
665     fprintf(file,"%d\n",qevent->event->xconfigurerequest.x);
666     fprintf(file,"%d\n",qevent->event->xconfigurerequest.y);
667     fprintf(file,"%d\n",qevent->event->xconfigurerequest.width);
668     fprintf(file,"%d\n",qevent->event->xconfigurerequest.height);
669     fprintf(file,"%d\n",qevent->event->xconfigurerequest.border_width);
670     fprintf(file,"%x\n",qevent->event->xconfigurerequest.above);
671     fprintf(file,"%d\n",qevent->event->xconfigurerequest.detail);
672     fprintf(file,"%d\n\n",qevent->event->xconfigurerequest.value_mask);
673   }
674
675   /*************************************************************************
676    * void StoreGravityNotify()
677    *************************************************************************/
678   void StoreGravityNotify(file, qevent)
679   FILE *file;
680   QEvent *qevent;
681   {
682     fprintf(file,"%d\n",qevent->event->xgravity.type);
683     fprintf(file,"%s\n",qevent->name);
684     fprintf(file,"%lu\n",qevent->time);
685     fprintf(file,"%x\n",qevent->event->xgravity.event);
686     fprintf(file,"%x\n",qevent->event->xgravity.window);
687     fprintf(file,"%d\n",qevent->event->xgravity.x);
688     fprintf(file,"%d\n\n",qevent->event->xgravity.y);
689   }
690
691   /*************************************************************************
692    * void StoreResizeRequest()
693    *************************************************************************/
694   void StoreResizeRequest(file, qevent)
695   FILE *file;
696   QEvent *qevent;
697   {
698     fprintf(file,"%d\n",qevent->event->xresizerequest.type);
699     fprintf(file,"%s\n",qevent->name);
700     fprintf(file,"%lu\n",qevent->time);
701     fprintf(file,"%x\n",qevent->event->xresizerequest.window);
702     fprintf(file,"%d\n",qevent->event->xresizerequest.width);
703     fprintf(file,"%d\n\n",qevent->event->xresizerequest.height);
704   }
705
706   /*************************************************************************
707    * void StoreCirculateNotify()
708    *************************************************************************/
709   void StoreCirculateNotify(file, qevent)
710   FILE *file;
711   QEvent *qevent;
712   {
713     fprintf(file,"%d\n",qevent->event->xcirculate.type);
714     fprintf(file,"%s\n",qevent->name);
715     fprintf(file,"%lu\n",qevent->time);
716     fprintf(file,"%x\n",qevent->event->xcirculate.event);
717     fprintf(file,"%x\n",qevent->event->xcirculate.window);
718     fprintf(file,"%d\n\n",qevent->event->xcirculate.place);
719   }
720
721   /*************************************************************************
722    * void StoreCirculateRequest()
723    *************************************************************************/
724   void StoreCirculateRequest(file, qevent)
725   FILE *file;
726   QEvent *qevent;
727   {
728     fprintf(file,"%d\n",qevent->event->xcirculaterequest.type);
729     fprintf(file,"%s\n",qevent->name);
730     fprintf(file,"%lu\n",qevent->time);
731     fprintf(file,"%x\n",qevent->event->xcirculaterequest.parent);
732     fprintf(file,"%x\n",qevent->event->xcirculaterequest.window);
733     fprintf(file,"%d\n\n",qevent->event->xcirculaterequest.place);
734   }
735
736   /*************************************************************************
737    * void StorePropertyNotify()
738    *************************************************************************/
739   void StorePropertyNotify(file, qevent)
740   FILE *file;
741   QEvent *qevent;
742   {
743     fprintf(file,"%d\n",qevent->event->xproperty.type);
744     fprintf(file,"%s\n",qevent->name);
745     fprintf(file,"%lu\n",qevent->time);
746     fprintf(file,"%x\n",qevent->event->xproperty.window);
747     fprintf(file,"%x\n",qevent->event->xproperty.atom);
748     fprintf(file,"%d\n",qevent->event->xproperty.time);
749     fprintf(file,"%d\n\n",qevent->event->xproperty.state);
750   }
751
752   /*************************************************************************
753    * void StoreSelectionClear()
754    *************************************************************************/
755   void StoreSelectionClear(file, qevent)
756   FILE *file;
757   QEvent *qevent;
758   {
759     fprintf(file,"%d\n",qevent->event->xselectionclear.type);
760     fprintf(file,"%s\n",qevent->name);
761     fprintf(file,"%lu\n",qevent->time);
762     fprintf(file,"%x\n",qevent->event->xselectionclear.window);
763     fprintf(file,"%x\n",qevent->event->xselectionclear.selection);
764     fprintf(file,"%d\n\n",qevent->event->xselectionclear.time);
765   }
766
767   /*************************************************************************
768    * void StoreSelectionRequest()
```

```
769  ************************************************************************/
770  void StoreSelectionRequest(file, qevent)
771  FILE *file;
772  QEvent *qevent;
773  {
774    fprintf(file,"%d\n",qevent->event->xselectionrequest.type);
775    fprintf(file,"%s\n",qevent->name);
776    fprintf(file,"%lu\n",qevent->time);
777    fprintf(file,"%x\n",qevent->event->xselectionrequest.owner);
778    fprintf(file,"%x\n",qevent->event->xselectionrequest.requestor);
779    fprintf(file,"%x\n",qevent->event->xselectionrequest.selection);
780    fprintf(file,"%x\n",qevent->event->xselectionrequest.target);
781    fprintf(file,"%x\n",qevent->event->xselectionrequest.property);
782    fprintf(file,"%d\n\n",qevent->event->xselectionrequest.time);
783  }
784
785  /************************************************************************
786   * void StoreSelectionNotify()
787   ************************************************************************/
788  void StoreSelectionNotify(file, qevent)
789  FILE *file;
790  QEvent *qevent;
791  {
792    fprintf(file,"%d\n",qevent->event->xselection.type);
793    fprintf(file,"%s\n",qevent->name);
794    fprintf(file,"%lu\n",qevent->time);
795    fprintf(file,"%x\n",qevent->event->xselection.requestor);
796    fprintf(file,"%x\n",qevent->event->xselection.selection);
797    fprintf(file,"%x\n",qevent->event->xselection.target);
798    fprintf(file,"%x\n",qevent->event->xselection.property);
799    fprintf(file,"%d\n\n",qevent->event->xselection.time);
800  }
801
802  /************************************************************************
803   * void StoreColormapNotify()
804   ************************************************************************/
805  void StoreColormapNotify(file, qevent)
806  FILE *file;
807  QEvent *qevent;
808  {
809    fprintf(file,"%d\n",qevent->event->xcolormap.type);
810    fprintf(file,"%s\n",qevent->name);
811    fprintf(file,"%lu\n",qevent->time);
812    fprintf(file,"%x\n",qevent->event->xcolormap.window);
813    fprintf(file,"%x\n",qevent->event->xcolormap.colormap);
814    fprintf(file,"%d\n",qevent->event->xcolormap.new);
815    fprintf(file,"%d\n\n",qevent->event->xcolormap.state);
816  }
817
818  /************************************************************************
819   * void StoreClientMessage()
820   ************************************************************************/
821  void StoreClientMessage(file, qevent)
822  FILE *file;
823  QEvent *qevent;
824  {
825    int i;
826
827    fprintf(file,"%d\n",qevent->event->xclient.type);
828    fprintf(file,"%s\n",qevent->name);
829    fprintf(file,"%lu\n",qevent->time);
830    fprintf(file,"%x\n",qevent->event->xclient.window);
831    fprintf(file,"%x\n",qevent->event->xclient.message_type);
832    fprintf(file,"%sn",qevent->name); fprintf(file,"%lun",qevent->time);
833    fprintf(file,"%d\n",qevent->event->xclient.format);
834
835    for (i=0; i<20; i++)
836      fprintf(file,"%x\n",qevent->event->xclient.data.b[i]);
837    fprintf(file,"\n");
838  }
839
840  /************************************************************************
841   * void StoreMappingNotify()
842   ************************************************************************/
843  void StoreMappingNotify(file, qevent)
844  FILE *file;
845  QEvent *qevent;
846  {
847    fprintf(file,"%d\n",qevent->event->xmapping.type);
848    fprintf(file,"%s\n",qevent->name);
849    fprintf(file,"%lu\n",qevent->time);
850    fprintf(file,"%x\n",qevent->event->xmapping.window);
851    fprintf(file,"%d\n",qevent->event->xmapping.request);
852    fprintf(file,"%d\n",qevent->event->xmapping.first_keycode);
853    fprintf(file,"%d\n\n",qevent->event->xmapping.count);
854  }
855
856  /************************************************************************
857   * Read Routines (file to event structure)
858   ************************************************************************/
859
860  /************************************************************************
861   * void ReadKeyPress()
862   ************************************************************************/
863  void ReadKeyPress(file,qevent)
864  FILE *file;
865  QEvent *qevent;
866  {
867    qevent->event->xkey.type = qevent->event->type;
868    fscanf(file,"%s",qevent->name);
869    fscanf(file,"%lu",&(qevent->time));
870    fscanf(file,"%x",&(qevent->event->xkey.window));
871    fscanf(file,"%x",&(qevent->event->xkey.root));
872    fscanf(file,"%x",&(qevent->event->xkey.subwindow));
873    fscanf(file,"%d",&(qevent->event->xkey.time));
874    fscanf(file,"%d",&(qevent->event->xkey.x));
875    fscanf(file,"%d",&(qevent->event->xkey.y));
876    fscanf(file,"%d",&(qevent->event->xkey.x_root));
877    fscanf(file,"%d",&(qevent->event->xkey.y_root));
878    fscanf(file,"%d",&(qevent->event->xkey.state));
879    fscanf(file,"%d",&(qevent->event->xkey.keycode));
880    fscanf(file,"%d",&(qevent->event->xkey.same_screen));
881  }
882
883  /************************************************************************
884   * void ReadKeyRelease()
885   ************************************************************************/
886  void ReadKeyRelease(file, qevent)
887  FILE *file;
888  QEvent *qevent;
889  {
890    /* It has the same info */
891    ReadKeyPress(file,qevent);
892  }
893
894  /************************************************************************
895   * void ReadButtonPress()
896   ************************************************************************/
```

```
897  void ReadButtonPress(file, qevent)
898  FILE *file;
899  QEvent *qevent;
900  {
901    qevent->event->xbutton.type = qevent->event->type;
902    fscanf(file,"%s",qevent->name);
903    fscanf(file,"%lu",&(qevent->time));
904    fscanf(file,"%x",&(qevent->event->xbutton.window));
905    fscanf(file,"%x",&(qevent->event->xbutton.root));
906    fscanf(file,"%x",&(qevent->event->xbutton.subwindow));
907    fscanf(file,"%d",&(qevent->event->xbutton.time));
908    fscanf(file,"%d",&(qevent->event->xbutton.x));
909    fscanf(file,"%d",&(qevent->event->xbutton.y));
910    fscanf(file,"%d",&(qevent->event->xbutton.x_root));
911    fscanf(file,"%d",&(qevent->event->xbutton.y_root));
912    fscanf(file,"%d",&(qevent->event->xbutton.state));
913    fscanf(file,"%d",&(qevent->event->xbutton.button));
914    fscanf(file,"%d",&(qevent->event->xbutton.same_screen));
915  }
916
917  /*************************************************************************
918   * void ReadButtonRelease()
919   *************************************************************************/
920  void ReadButtonRelease(file, qevent)
921  FILE *file;
922  QEvent *qevent;
923  {
924    /* It has the same info */
925    ReadButtonPress(file,qevent);
926  }
927
928  /*************************************************************************
929   * void ReadMotionNotify()
930   *************************************************************************/
931  void ReadMotionNotify(file, qevent)
932  FILE *file;
933  QEvent *qevent;
934  {
935    qevent->event->xmotion.type = qevent->event->type;
936    fscanf(file,"%s",qevent->name);
937    fscanf(file,"%lu",&(qevent->time));
938    fscanf(file,"%x\n",&(qevent->event->xmotion.window));
939    fscanf(file,"%x\n",&(qevent->event->xmotion.root));
940    fscanf(file,"%x\n",&(qevent->event->xmotion.subwindow));
941    fscanf(file,"%d\n",&(qevent->event->xmotion.time));
942    fscanf(file,"%d\n",&(qevent->event->xmotion.x));
943    fscanf(file,"%d\n",&(qevent->event->xmotion.y));
944    fscanf(file,"%d\n",&(qevent->event->xmotion.x_root));
945    fscanf(file,"%d\n",&(qevent->event->xmotion.y_root));
946    fscanf(file,"%d\n",&(qevent->event->xmotion.state));
947    fscanf(file,"%c\n",&(qevent->event->xmotion.is_hint));
948    fscanf(file,"%d\n\n",&(qevent->event->xmotion.same_screen));
949  }
950
951  /*************************************************************************
952   * void ReadEnterNotify()
953   *************************************************************************/
954  void ReadEnterNotify(file, qevent)
955  FILE *file;
956  QEvent *qevent;
957  {
958    qevent->event->xcrossing.type = qevent->event->type;
959    fscanf(file,"%s",qevent->name);
960    fscanf(file,"%lu",&(qevent->time));
961    fscanf(file,"%x",&(qevent->event->xcrossing.window));
962    fscanf(file,"%x",&(qevent->event->xcrossing.root));
963    fscanf(file,"%x",&(qevent->event->xcrossing.subwindow));
964    fscanf(file,"%d",&(qevent->event->xcrossing.time));
965    fscanf(file,"%d",&(qevent->event->xcrossing.x));
966    fscanf(file,"%d",&(qevent->event->xcrossing.y));
967    fscanf(file,"%d",&(qevent->event->xcrossing.x_root));
968    fscanf(file,"%d",&(qevent->event->xcrossing.y_root));
969    fscanf(file,"%d",&(qevent->event->xcrossing.mode));
970    fscanf(file,"%d",&(qevent->event->xcrossing.detail));
971    fscanf(file,"%d",&(qevent->event->xcrossing.same_screen));
972    fscanf(file,"%d",&(qevent->event->xcrossing.focus));
973    fscanf(file,"%d",&(qevent->event->xcrossing.state));
974
975  }
976
977  /*************************************************************************
978   * void ReadLeaveNotify()
979   *************************************************************************/
980  void ReadLeaveNotify(file, qevent)
981  FILE *file;
982  QEvent *qevent;
983  {
984    /* It has the same info */
985    ReadEnterNotify(file,qevent);
986  }
987
988  /*************************************************************************
989   * void ReadFocusIn()
990   *************************************************************************/
991  void ReadFocusIn(file, qevent)
992  FILE *file;
993  QEvent *qevent;
994  {
995    qevent->event->xfocus.type = qevent->event->type;
996    fscanf(file,"%s",qevent->name);
997    fscanf(file,"%lu",&(qevent->time));
998    fscanf(file,"%x",&(qevent->event->xfocus.window));
999    fscanf(file,"%d",&(qevent->event->xfocus.mode));
1000   fscanf(file,"%d",&(qevent->event->xfocus.detail));
1001 }
1002
1003 /*************************************************************************
1004  * void ReadFocusOut()
1005  *************************************************************************/
1006 void ReadFocusOut(file, qevent)
1007 FILE *file;
1008 QEvent *qevent;
1009 {
1010   /* It has the same info */
1011   ReadFocusIn(file,qevent);
1012 }
1013
1014 /*************************************************************************
1015  * void ReadKeymapNotify()
1016  *************************************************************************/
1017 void ReadKeymapNotify(file, qevent)
1018 FILE *file;
1019 QEvent *qevent;
1020 {
1021   int i;
1022
1023   qevent->event->xkeymap.type = qevent->event->type;
1024   fscanf(file,"%s",qevent->name);
```

```
1025    fscanf(file,"%lu",&(qevent->time));
1026    fscanf(file,"%x\n",&(qevent->event->xkeymap.window));
1027
1028    for (i=0; i<32; i++)
1029      fscanf(file,"%x\n",&(qevent->event->xkeymap.key_vector[i]));
1030  }
1031
1032  /*************************************************************************
1033   * void ReadExpose()
1034   *************************************************************************/
1035  void ReadExpose(file, qevent)
1036  FILE *file;
1037  QEvent *qevent;
1038  {
1039    qevent->event->xexpose.type = qevent->event->type;
1040    fscanf(file,"%s",qevent->name);
1041    fscanf(file,"%lu",&(qevent->time));
1042    fscanf(file,"%x\n",&(qevent->event->xexpose.window));
1043    fscanf(file,"%d\n",&(qevent->event->xexpose.x));
1044    fscanf(file,"%d\n",&(qevent->event->xexpose.y));
1045    fscanf(file,"%d\n",&(qevent->event->xexpose.width));
1046    fscanf(file,"%d\n",&(qevent->event->xexpose.height));
1047    fscanf(file,"%d\n\n",&(qevent->event->xexpose.count));
1048  }
1049
1050  /*************************************************************************
1051   * void ReadGraphicsExpose()
1052   *************************************************************************/
1053  void ReadGraphicsExpose(file, qevent)
1054  FILE *file;
1055  QEvent *qevent;
1056  {
1057    qevent->event->xgraphicsexpose.type = qevent->event->type;
1058    fscanf(file,"%s",qevent->name);
1059    fscanf(file,"%lu",&(qevent->time));
1060    fscanf(file,"%x\n",&(qevent->event->xgraphicsexpose.drawable));
1061    fscanf(file,"%d\n",&(qevent->event->xgraphicsexpose.x));
1062    fscanf(file,"%d\n",&(qevent->event->xgraphicsexpose.y));
1063    fscanf(file,"%d\n",&(qevent->event->xgraphicsexpose.width));
1064    fscanf(file,"%d\n",&(qevent->event->xgraphicsexpose.height));
1065    fscanf(file,"%d\n",&(qevent->event->xgraphicsexpose.count));
1066    fscanf(file,"%d\n",&(qevent->event->xgraphicsexpose.major_code));
1067    fscanf(file,"%d\n\n",&(qevent->event->xgraphicsexpose.minor_code));
1068  }
1069
1070  /*************************************************************************
1071   * void ReadNoExpose()
1072   *************************************************************************/
1073  void ReadNoExpose(file, qevent)
1074  FILE *file;
1075  QEvent *qevent;
1076  {
1077    qevent->event->xnoexpose.type = qevent->event->type;
1078    fscanf(file,"%s",qevent->name);
1079    fscanf(file,"%lu",&(qevent->time));
1080    fscanf(file,"%x\n",&(qevent->event->xnoexpose.drawable));
1081    fscanf(file,"%d\n",&(qevent->event->xnoexpose.major_code));
1082    fscanf(file,"%d\n\n",&(qevent->event->xnoexpose.minor_code));
1083  }
1084
1085  /*************************************************************************
1086   * void ReadVisibilityNotify()
1087   *************************************************************************/
1088  void ReadVisibilityNotify(file, qevent)
1089  FILE *file;
1090  QEvent *qevent;
1091  {
1092    qevent->event->xvisibility.type = qevent->event->type;
1093    fscanf(file,"%s",qevent->name);
1094    fscanf(file,"%lu",&(qevent->time));
1095    fscanf(file,"%x\n",&(qevent->event->xvisibility.window));
1096    fscanf(file,"%d\n\n",&(qevent->event->xvisibility.state));
1097  }
1098
1099  /*************************************************************************
1100   * void ReadCreateNotify()
1101   *************************************************************************/
1102  void ReadCreateNotify(file, qevent)
1103  FILE *file;
1104  QEvent *qevent;
1105  {
1106    qevent->event->xcreatewindow.type = qevent->event->type;
1107    fscanf(file,"%s",qevent->name);
1108    fscanf(file,"%lu",&(qevent->time));
1109    fscanf(file,"%x\n",&(qevent->event->xcreatewindow.parent));
1110    fscanf(file,"%x\n",&(qevent->event->xcreatewindow.window));
1111    fscanf(file,"%d\n",&(qevent->event->xcreatewindow.x));
1112    fscanf(file,"%d\n",&(qevent->event->xcreatewindow.y));
1113    fscanf(file,"%d\n",&(qevent->event->xcreatewindow.width));
1114    fscanf(file,"%d\n",&(qevent->event->xcreatewindow.height));
1115    fscanf(file,"%d\n\n",&(qevent->event->xcreatewindow.override_redirect));
1116  }
1117
1118  /*************************************************************************
1119   * void ReadDestroyNotify()
1120   *************************************************************************/
1121  void ReadDestroyNotify(file, qevent)
1122  FILE *file;
1123  QEvent *qevent;
1124  {
1125    qevent->event->xdestroywindow.type = qevent->event->type;
1126    fscanf(file,"%s",qevent->name);
1127    fscanf(file,"%lu",&(qevent->time));
1128    fscanf(file,"%x\n",&(qevent->event->xdestroywindow.event));
1129    fscanf(file,"%x\n\n",&(qevent->event->xdestroywindow.window));
1130  }
1131
1132  /*************************************************************************
1133   * void ReadUnmapNotify()
1134   *************************************************************************/
1135  void ReadUnmapNotify(file, qevent)
1136  FILE *file;
1137  QEvent *qevent;
1138  {
1139    qevent->event->xunmap.type = qevent->event->type;
1140    fscanf(file,"%s",qevent->name);
1141    fscanf(file,"%lu",&(qevent->time));
1142    fscanf(file,"%x\n",&(qevent->event->xunmap.event));
1143    fscanf(file,"%x\n",&(qevent->event->xunmap.window));
1144    fscanf(file,"%d\n\n",&(qevent->event->xunmap.from_configure));
1145  }
1146
1147  /*************************************************************************
1148   * void ReadMapNotify()
1149   *************************************************************************/
1150  void ReadMapNotify(file, qevent)
1151  FILE *file;
1152  QEvent *qevent;
```

```
1153  {
1154    qevent->event->xmap.type = qevent->event->type;
1155    fscanf(file,"%s",qevent->name);
1156    fscanf(file,"%lu",&(qevent->time));
1157    fscanf(file,"%x\n",&(qevent->event->xmap.event));
1158    fscanf(file,"%x\n",&(qevent->event->xmap.window));
1159    fscanf(file,"%d\n\n",&(qevent->event->xmap.override_redirect));
1160  }
1161
1162  /*************************************************************************
1163   * void ReadMapRequest()
1164   *************************************************************************/
1165  void ReadMapRequest(file, qevent)
1166  FILE *file;
1167  QEvent *qevent;
1168  {
1169    qevent->event->xmaprequest.type = qevent->event->type;
1170    fscanf(file,"%s",qevent->name);
1171    fscanf(file,"%lu",&(qevent->time));
1172    fscanf(file,"%x\n",&(qevent->event->xmaprequest.parent));
1173    fscanf(file,"%x\n\n",&(qevent->event->xmaprequest.window));
1174  }
1175
1176  /*************************************************************************
1177   * void ReadReparentNotify()
1178   *************************************************************************/
1179  void ReadReparentNotify(file, qevent)
1180  FILE *file;
1181  QEvent *qevent;
1182  {
1183    qevent->event->xreparent.type = qevent->event->type;
1184    fscanf(file,"%s",qevent->name);
1185    fscanf(file,"%lu",&(qevent->time));
1186    fscanf(file,"%x\n",&(qevent->event->xreparent.event));
1187    fscanf(file,"%x\n",&(qevent->event->xreparent.window));
1188    fscanf(file,"%x\n",&(qevent->event->xreparent.parent));
1189    fscanf(file,"%d\n",&(qevent->event->xreparent.x));
1190    fscanf(file,"%d\n",&(qevent->event->xreparent.y));
1191    fscanf(file,"%d\n\n",&(qevent->event->xreparent.override_redirect));
1192  }
1193
1194  /*************************************************************************
1195   * void ReadConfigureNotify()
1196   *************************************************************************/
1197  void ReadConfigureNotify(file, qevent)
1198  FILE *file;
1199  QEvent *qevent;
1200  {
1201    qevent->event->xconfigure.type = qevent->event->type;
1202    fscanf(file,"%s",qevent->name);
1203    fscanf(file,"%lu",&(qevent->time));
1204    fscanf(file,"%x\n",&(qevent->event->xconfigure.event));
1205    fscanf(file,"%x\n",&(qevent->event->xconfigure.window));
1206    fscanf(file,"%d\n",&(qevent->event->xconfigure.x));
1207    fscanf(file,"%d\n",&(qevent->event->xconfigure.y));
1208    fscanf(file,"%d\n",&(qevent->event->xconfigure.width));
1209    fscanf(file,"%d\n",&(qevent->event->xconfigure.height));
1210    fscanf(file,"%d\n",&(qevent->event->xconfigure.border_width));
1211    fscanf(file,"%x\n",&(qevent->event->xconfigure.above));
1212    fscanf(file,"%d\n\n",&(qevent->event->xconfigure.override_redirect));
1213  }
1214
1215  /*************************************************************************
1216   * void ReadConfigureRequest()
1217   *************************************************************************/
1218  void ReadConfigureRequest(file, qevent)
1219  FILE *file;
1220  QEvent *qevent;
1221  {
1222    qevent->event->xconfigurerequest.type = qevent->event->type;
1223    fscanf(file,"%s",qevent->name);
1224    fscanf(file,"%lu",&(qevent->time));
1225    fscanf(file,"%x\n",&(qevent->event->xconfigurerequest.parent));
1226    fscanf(file,"%x\n",&(qevent->event->xconfigurerequest.window));
1227    fscanf(file,"%d\n",&(qevent->event->xconfigurerequest.x));
1228    fscanf(file,"%d\n",&(qevent->event->xconfigurerequest.y));
1229    fscanf(file,"%d\n",&(qevent->event->xconfigurerequest.width));
1230    fscanf(file,"%d\n",&(qevent->event->xconfigurerequest.height));
1231    fscanf(file,"%d\n",&(qevent->event->xconfigurerequest.border_width));
1232    fscanf(file,"%x\n",&(qevent->event->xconfigurerequest.above));
1233    fscanf(file,"%d\n",&(qevent->event->xconfigurerequest.detail));
1234    fscanf(file,"%d\n\n",&(qevent->event->xconfigurerequest.value_mask));
1235  }
1236
1237  /*************************************************************************
1238   * void ReadGravityNotify()
1239   *************************************************************************/
1240  void ReadGravityNotify(file, qevent)
1241  FILE *file;
1242  QEvent *qevent;
1243  {
1244    qevent->event->xgravity.type = qevent->event->type;
1245    fscanf(file,"%s",qevent->name);
1246    fscanf(file,"%lu",&(qevent->time));
1247    fscanf(file,"%x\n",&(qevent->event->xgravity.event));
1248    fscanf(file,"%x\n",&(qevent->event->xgravity.window));
1249    fscanf(file,"%d\n",&(qevent->event->xgravity.x));
1250    fscanf(file,"%d\n\n",&(qevent->event->xgravity.y));
1251  }
1252
1253  /*************************************************************************
1254   * void ReadResizeRequest()
1255   *************************************************************************/
1256  void ReadResizeRequest(file, qevent)
1257  FILE *file;
1258  QEvent *qevent;
1259  {
1260    qevent->event->xresizerequest.type = qevent->event->type;
1261    fscanf(file,"%s",qevent->name);
1262    fscanf(file,"%lu",&(qevent->time));
1263    fscanf(file,"%x\n",&(qevent->event->xresizerequest.window));
1264    fscanf(file,"%d\n",&(qevent->event->xresizerequest.width));
1265    fscanf(file,"%d\n\n",&(qevent->event->xresizerequest.height));
1266  }
1267
1268  /*************************************************************************
1269   * void ReadCirculateNotify()
1270   *************************************************************************/
1271  void ReadCirculateNotify(file, qevent)
1272  FILE *file;
1273  QEvent *qevent;
1274  {
1275    qevent->event->xcirculate.type = qevent->event->type;
1276    fscanf(file,"%s",qevent->name);
1277    fscanf(file,"%lu",&(qevent->time));
1278    fscanf(file,"%x\n",&(qevent->event->xcirculate.event));
1279    fscanf(file,"%x\n",&(qevent->event->xcirculate.window));
1280    fscanf(file,"%d\n\n",&(qevent->event->xcirculate.place));
```

```
}

/*************************************************************************
 * void ReadCirculateRequest()
 *************************************************************************/
void ReadCirculateRequest(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xcirculaterequest.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xcirculaterequest.parent));
  fscanf(file,"%x\n",&(qevent->event->xcirculaterequest.window));
  fscanf(file,"%d\n\n",&(qevent->event->xcirculaterequest.place));
}
/*************************************************************************
 * void ReadPropertyNotify()
 *************************************************************************/
void ReadPropertyNotify(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xproperty.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xproperty.window));
  fscanf(file,"%x\n",&(qevent->event->xproperty.atom));
  fscanf(file,"%d\n",&(qevent->event->xproperty.time));
  fscanf(file,"%d\n\n",&(qevent->event->xproperty.state));
}

/*************************************************************************
 * void ReadSelectionClear()
 *************************************************************************/
void ReadSelectionClear(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xselectionclear.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xselectionclear.window));
  fscanf(file,"%x\n",&(qevent->event->xselectionclear.selection));
  fscanf(file,"%d\n\n",&(qevent->event->xselectionclear.time));
}

/*************************************************************************
 * void ReadSelectionRequest()
 *************************************************************************/
void ReadSelectionRequest(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xselectionrequest.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xselectionrequest.owner));
  fscanf(file,"%x\n",&(qevent->event->xselectionrequest.requestor));
  fscanf(file,"%x\n",&(qevent->event->xselectionrequest.selection));
  fscanf(file,"%x\n",&(qevent->event->xselectionrequest.target));
  fscanf(file,"%x\n",&(qevent->event->xselectionrequest.property));
  fscanf(file,"%d\n\n",&(qevent->event->xselectionrequest.time));
}

/*************************************************************************
 * void ReadSelectionNotify()
 *************************************************************************/
void ReadSelectionNotify(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xselection.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xselection.requestor));
  fscanf(file,"%x\n",&(qevent->event->xselection.selection));
  fscanf(file,"%x\n",&(qevent->event->xselection.target));
  fscanf(file,"%x\n",&(qevent->event->xselection.property));
  fscanf(file,"%d\n\n",&(qevent->event->xselection.time));
}

/*************************************************************************
 * void ReadColormapNotify()
 *************************************************************************/
void ReadColormapNotify(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xcolormap.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xcolormap.window));
  fscanf(file,"%x\n",&(qevent->event->xcolormap.colormap));
  fscanf(file,"%d\n",&(qevent->event->xcolormap.new));
  fscanf(file,"%d\n\n",&(qevent->event->xcolormap.state));
}

/*************************************************************************
 * void ReadClientMessage()
 *************************************************************************/
void ReadClientMessage(file, qevent)
FILE *file;
QEvent *qevent;
{
  int i;

  qevent->event->xclient.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
  fscanf(file,"%x\n",&(qevent->event->xclient.window));
  fscanf(file,"%x\n",&(qevent->event->xclient.message_type));
  fscanf(file,"%d\n",&(qevent->event->xclient.format));

  for (i=0; i<20; i++)
    fscanf(file,"%x\n",&(qevent->event->xclient.data.b[i]));
}

/*************************************************************************
 * void ReadMappingNotify()
 *************************************************************************/
void ReadMappingNotify(file, qevent)
FILE *file;
QEvent *qevent;
{
  qevent->event->xmapping.type = qevent->event->type;
  fscanf(file,"%s",qevent->name);
  fscanf(file,"%lu",&(qevent->time));
```

```
1409    fscanf(file,"%x\n",&(qevent->event->xmapping.window));
1410    fscanf(file,"%d\n",&(qevent->event->xmapping.request));
1411    fscanf(file,"%d\n",&(qevent->event->xmapping.first_keycode));
1412    fscanf(file,"%d\n\n",&(qevent->event->xmapping.count));
1413  }
```

```
 ***************************************************************************
 * queue.h
 ***************************************************************************

#include <X11/Intrinsic.h>


#ifndef _QUEUE_H_
#define _QUEUE_H_

/*
 * QEvent is the data structure that stores the events
 * coming from the CM. It stores thme in memory.
 */
typedef struct _QEvent {
    struct _QEvent *next;
    XEvent *event;
    char *name;
    long time;
} QEvent;


typedef struct{
  QEvent *front;
  QEvent *rear;
  QEvent *prev;
} EventQueue;


/***************************************************************************
 Prototypes
 ***************************************************************************/
QEvent *AllocNode();     /* Allocates a node (memory) for the queue */
void   InitQueue();      /* Initializes queue pointers */
QEvent *CreateNode();    /* Creates a new node with the info passed */
void   EnQueue();        /* Enqueues at end of queue */
QEvent *DeQueue();       /* Takes front node from the queue */
void   FreeQueue();      /* Deallocates memory for the whole queue */
void   DeleteNode();     /* Deallocates memory for a node */
void   AddName();        /* Adds name to the last node in the queue */
void   AddTime();        /* Adds time to the last node in the queue */
void   PrintQueue();     /* Prints contents of queue */

#endif
```

```
/*************************************************************************
 * queue.c
 *************************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "XemsEvent.h"
#include "queue.h"
#include "util.h"

#include "ui.h"

/*************************************************************************
 * QEvent *AllocNode()
 *
 * Input:
 * Output:
 * Algo:        Allocates space for queue node
 * Side Effects:
 *
 *************************************************************************/
QEvent *AllocNode()
{
  QEvent *n;
  n = XtNew(QEvent);
  n->next = NULL;
  n->name = NULL;

  return n;
}


/*************************************************************************
 * void InitQueue(q)
 * Input:
 * Output:
 * Algo:        Allocates space for queue node
 * Side Effects:
 *
 *************************************************************************/
void InitQueue(q)
EventQueue *q;
{
  q->front = NULL;
  q->rear  = NULL;
  q->prev  = NULL;
}

/*************************************************************************
 * QEvent *CreateNode(e)
 * Input:
 * Output:
 * Algo:        Allocates space for queue node
 * Side Effects:
 *
 *************************************************************************/
QEvent *CreateNode(e)
XEvent *e;
{
  XEvent *p;
  QEvent *n;
  n = AllocNode();

  n->event = XtNew(XEvent);
  bcopy(e,n->event,sizeof(*e));

  return n;
}


/*************************************************************************
 * void EnQueue(q,p)
 * Adds node 'p' to queue 'q'
 * Input:
 * Output:
 * Algo:        Allocates space for queue node
 * Side Effects:
 *
 *************************************************************************/
void EnQueue(q,p) /* adds node 'p' to queue 'q' */
EventQueue *q;
QEvent *p;
{
  if (q->front == NULL) /* first element in queue*/
    {
      q->front = p;
      q->rear = q->front;
    }
  else /* queue already has something */
    {
      q->rear->next = p;        /* attach new node */
      q->prev = q->rear;       /* "remember" last node */
      q->rear = q->rear->next;  /* point to last node */
    }
}

/*************************************************************************
 * QEvent *DeQueue(q)
 * Input:
 * Output:
 * Algo:        Allocates space for queue node
 * Side Effects:
 *
 *************************************************************************/
QEvent *DeQueue(q)
EventQueue *q;
{
  QEvent *t;

  t = q->front;
  if (t != NULL)
    {
      q->front = q->front->next;
      t->next = NULL;
    }
  return t;
}

/*************************************************************************
 * void FreeQueue(q)
 * Input:
 * Output:
 * Algo:        Allocates space for queue node
 * Side Effects:
 *
 *************************************************************************/
```

```
129 void FreeQueue(q)
130 EventQueue *q;
131 {
132   QEvent *t,*p;
133
134   p = t = q->front;
135   while (t != NULL)
136     {
137       XtFree(t->event);
138       XtFree(t->name);
139       p = t->next;
140       XtFree(t);
141       t = p;
142     }
143   InitQueue(q);
144 }
145
146 /*************************************************************************
147  * void AddName(q,nm)
148  * Input:
149  * Output:
150  * Algo:        Allocates space for queue node
151  * Side Effects:
152  *
153  *************************************************************************/
154 void AddName(q,nm)
155 EventQueue *q;
156 char *nm;
157 {
158   if (q->rear)
159     {
160       q->rear->name = XtMalloc(strlen(nm)+1);
161       strcpy(q->rear->name, nm);
162       q->rear->name[strlen(nm)+1] = '\0';
163     }
164 }
165
166 /*************************************************************************
167  * void AddTime(q, t)
168  * Input:
169  * Output:
170  * Algo:        Allocates space for queue node
171  * Side Effects:
172  *
173  *************************************************************************/
174 void AddTime(q, t)
175 EventQueue *q;
176 long t;
177 {
178   if (q->rear) /* if queue has anything */
179     {
180       q->rear->time = t;
181     }
182 }
183
184
185
186 /*************************************************************************
187  * void PrintQueue(q)
188  * Input:
189  * Output:
190  * Algo:        Allocates space for queue node
191  * Side Effects:
192  *
193  *************************************************************************/
194 void PrintQueue(q)
195 EventQueue *q;
196 {
197   QEvent *p;
198   p = q->front;
199
200   if (p == NULL)
201     MessagePrint("Queue is empty!\n");
202
203   while (p!=NULL)
204     {
205       printf("1-----------------------------------\n");
206       PrintEvent(p->event);
207       printf("Name: %s\n", p->name);
208       printf("Time: %lu\n", p->time);
209       printf("2-----------------------------------\n");
210       p = p->next;
211     }
212 }
```

```
1  /*****************************************************************************
2   * ui.h
3   *
4   *****************************************************************************
5
6  #include <X11/Intrinsic.h>
7
8
9  /* Menu structure */
10 typedef struct _menu_item
11 {
12   char        *label;          /* the label for the item */
13   WidgetClass *class;          /* pushbutton, label, separator... */
14   char        mnemonic;        /* mnemonic; NULL if none */
15   char        *accelerator;    /* accelerator; NULL if none */
16   char        *accel_text;     /* to be converted to compound string */
17   void        (*callback)();   /* routine to call; NULL if none */
18   XtPointer   callback_data;   /* client_data for callback() */
19   struct _menu_item *subitems; /* pullright menu items, if not NULL */
20 } MenuItem;
21
22
23 /* Menu builder */
24 Widget BuildPulldownMenu();
25
26 /* Setup Panels */
27 void SetupStatePanel();
28 void SetupActionPanel();
29 void SetupActionPanel();
30 void SetupFilterPanel();
31
32 /* uiMessage */
33 void MessagePrint();
34
35 /* File Callbacks */
36 void New_CB();
37 void Save_CB();
38 void Retrieve_CB();
39 void Quit_CB();
40
41 /* State Callbacks*/
42 void Normal_CB();
43 void Record_CB();
44 void Play_CB();
45
46 /* Queue Callbacks */
47 void QPrint_CB();
48 void QSend_CB();
49 void QStep_CB();
50 void QReset_CB();
51 void QStop_CB();
52 void QSendNext_CB();
53
54 void MessageOn_CB();
55 void MessageOff_CB();
56
57 void FilterPop_CB();
58 void FilterDown_CB();
59 void ButtonToggle_CB();
60
61 /* Select Callbaks */
62 void SelectWindow_CB();
63 void SelectPrintWin();
64 void SelectPrintName();
65
66 void toggled();
```

```
1   /*************************************************************************
2    * ui.c
3    *
4    *************************************************************************/
5   #include <stdio.h>
6
7   #include <Xm/RowColumn.h> /* Motif stuff */
8   #include <Xm/MainW.h>
9   #include <Xm/PushB.h>
10  #include <Xm/PushBG.h>
11  #include <Xm/Form.h>
12  #include <Xm/ToggleBG.h>
13  #include <Xm/Frame.h>
14  #include <X11/Xatom.h>
15
16  #include "queue.h"
17  #include "ed.h"
18  #include "XemsEvent.h"
19  #include "file.h"
20  #include "ui.h"
21
22  extern GlobalInfo *EdInfo;
23  extern EventQueue Q;
24  QEvent *q_pointer = NULL;
25  long   f_m[2];
26  long   *filter_mask = f_m;
27  static XtIntervalId timer = NULL;
28  extern Bool print_flag;
29
30  /*************************************************************************
31   * File Callbacks
32   *************************************************************************/
33  /*************************************************************************
34   * Input:
35   * Output:
36   * Algo:
37   * Side Effects:
38   *************************************************************************/
39  void New_CB(w, client_data, call_data)
40  Widget w;
41  XtPointer client_data, call_data;
42  {
43    MessagePrint("New Queue\n");
44    FreeQueue(&Q);
45  }
46
47  /*************************************************************************
48   * Input:
49   * Output:
50   * Algo:
51   * Side Effects:
52   *************************************************************************/
53  void Save_CB(w, client_data, call_data)
54  Widget w;
55  XtPointer client_data, call_data;
56  {
57    MessagePrint("Saved file output.xes\n");
58    EdQueueToFile(&Q);
59  }
60
61  /*************************************************************************
62   * Input:
63   * Output:
64   * Algo:
65   * Side Effects:
66   *************************************************************************/
67  void Retrieve_CB(w, client_data, call_data)
68  Widget w;
69  XtPointer client_data, call_data;
70  {
71    MessagePrint("Retrieved file output.xes\n");
72    EdFileToQueue(&Q);
73  }
74
75  /*************************************************************************
76   * Input:
77   * Output:
78   * Algo:
79   * Side Effects:
80   *************************************************************************/
81  void Quit_CB(w, client_data, call_data)
82  Widget w;
83  XtPointer client_data, call_data;
84  {
85    MessagePrint("Quit\n");
86    exit(0);
87  }
88
89  /*************************************************************************
90   * Queue Callbacks
91   *************************************************************************/
92  /*************************************************************************
93   * Input:
94   * Output:
95   * Algo:
96   * Side Effects:
97   *************************************************************************/
98  void QPrint_CB(w, client_data, call_data)
99  Widget w;
100 XtPointer client_data, call_data;
101 {
102   MessagePrint("Printing Queue\n");
103   PrintQueue(&Q);
104 }
105
106 /*************************************************************************
107  * Input:
108  * Output:
109  * Algo:
110  * Side Effects:
111  *************************************************************************/
112 void QSend_CB(w, client_data, call_data)
113 Widget w;
114 XtPointer client_data, call_data;
115 {
116   if (EdInfo->cm_window != NULL)
117     {
118       MessagePrint("Sending Queue\n");
119       QSendNext_CB();
120     }
121   else
122     MessagePrint("Select Window First\n");
123 }
124
125 /*************************************************************************
126  * Input:
127  * Output:
128  * Algo:
```

```
 * Side Effects:
 ************************************************************************/
void QStep_CB(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  if (EdInfo->cm_window != NULL)
    {
      EdCheckValidWindow(EdInfo->cm_window);
      if (q_pointer == NULL)    /* first time */
        q_pointer = Q.front;

      if (q_pointer != NULL)
        {
          if (q_pointer->next != NULL)
            {
              MessagePrint("Stepping on Queue\n");
              EdSendEventQueue(q_pointer);
              q_pointer = q_pointer->next;
            }
        }
      else
        MessagePrint("Select Window First\n");
    }
}


/************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 ************************************************************************/
void QReset_CB(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  MessagePrint("Queue pointer reset to start\n");
  q_pointer = Q.front;
}


/************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 ************************************************************************/
void QStop_CB(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  MessagePrint("Queue Stop\n");
  if (timer != NULL)
    XtRemoveTimeOut(timer);
}


/************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 ************************************************************************/
void QSendNext_CB(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  EdCheckValidWindow(EdInfo->cm_window);

  if (q_pointer == NULL)    /* first time */
    q_pointer = Q.front;

  if (q_pointer != NULL)
    {
      if (q_pointer->next != NULL)
        {
          EdSendEventQueue(q_pointer);

          timer =
            XtAppAddTimeOut(XtWidgetToApplicationContext(
              XtWindowToWidget(EdInfo->ed_display, EdInfo->ed_window)),
                q_pointer->next->time - q_pointer->time,
                  QSendNext_CB, NULL);

          q_pointer = q_pointer->next;
        }
      else
        MessagePrint("End of Queue\n");
    }
}

/*************************************************************************
 * Message Callbacks
 *************************************************************************/

/************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 ************************************************************************/
void MessageOn_CB(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  print_flag = True;
  MessagePrint("Event Messages On\n");
}


/************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 ************************************************************************/
void MessageOff_CB(w, client_data, call_data)
Widget w;
XtPointer client_data, call_data;
{
  print_flag = False;
  MessagePrint("Event Messages Off\n");
}


/*************************************************************************
 * FilterCallbacks
```

```
257  ***************************************************************************
258  ***************************************************************************
259   * Input:
260   * Output:
261   * Algo:
262   * Side Effects:
263  ***************************************************************************
264  void FilterPop_CB(w, client_data, call_data)
265  Widget w;
266  XtPointer client_data, call_data;
267  {
268
269    XtPopup((Widget)client_data, XtGrabNone);
270
271  }
272
273
274  ***************************************************************************
275   * Input:
276   * Output:
277   * Algo:
278   * Side Effects:
279  ***************************************************************************
280  void FilterDown_CB(w, client_data, call_data)
281  Widget w;
282  XtPointer client_data, call_data;
283  {
284    XtPopdown((Widget)client_data);
285  }
286
287
288  ***************************************************************************
289   * Input:
290   * Output:
291   * Algo:
292   * Side Effects:
293  ***************************************************************************
294  void ButtonToggle_CB(w, type, state)
295  Widget w;
296  int type;
297  XmToggleButtonCallbackStruct *state;
298  {
299    printf("Button %d is %s\n", type, state->set? "on":"off");
300
301    if (type < 31)
302      filter_mask[0] = (filter_mask[0] & ~(1L<<type)) |
303        (((long)state->set)<<(type));
304    else
305      filter_mask[1] = (filter_mask[1] & ~(1L<<type)) |
306        (((long)state->set)<<type);
307
308    printf("Mask %lx\n", filter_mask[0]);
309    printf("Mask %lx\n", filter_mask[1]);
310
311  }
312
313
314  ***************************************************************************
315   * State Callbacks
316   ***************************************************************************
317  ***************************************************************************
318   * Input:
319   * Output:
320   * Algo:
321   * Side Effects:
322   ***************************************************************************
323  void Normal_CB(w, client_data, call_data)
324  Widget w;
325  XtPointer client_data, call_data;
326  {
327    if (EdSendMessage(EdInfo->ed_display, EdInfo->cm_window, NORMAL))
328      {
329        EdInfo->state = NORMAL;
330        MessagePrint("Sending Normal Message\n");
331      }
332  }
333
334
335  ***************************************************************************
336   * Input:
337   * Output:
338   * Algo:
339   * Side Effects:
340   ***************************************************************************
341  void Record_CB(w, client_data, call_data)
342  Widget w;
343  XtPointer client_data, call_data;
344  {
345    if (EdSendMessage(EdInfo->ed_display, EdInfo->cm_window, RECORD))
346      {
347        EdInfo->state = RECORD;
348        MessagePrint("Sending Record Message\n");
349      }
350  }
351
352
353  ***************************************************************************
354   * Input:
355   * Output:
356   * Algo:
357   * Side Effects:
358   ***************************************************************************
359  void Play_CB(w, client_data, call_data)
360  Widget w;
361  XtPointer client_data, call_data;
362  {
363    if (EdSendMessage(EdInfo->ed_display, EdInfo->cm_window, PLAY))
364      {
365        EdInfo->state = PLAY;
366        MessagePrint("Sending Play Message\n");
367      }
368  }
369
370  ***************************************************************************
371   * Select Callbacks
372   ***************************************************************************
373  ***************************************************************************
374   * Input:
375   * Output:
376   * Algo:
377   * Side Effects:
378   ***************************************************************************
379  void SelectWindow_CB(w, client_data, call_data)
380  Widget w;
381  XtPointer client_data, call_data;
382  {
383    Window win;
384
```

```
385     win = EdSelectWindow(EdInfo->ed_display);
386     win = XmuClientWindow(EdInfo->ed_display, win);
387     EdCheckValidWindow(win);
388
389     EdInfo->cm_window = win;
390
391     SelectPrintWin(client_data, win);
392     SelectPrintName(client_data, win);
393     MessagePrint("Selected Window\n");
394   }
395
396
397   /*************************************************************************
398    * Input:
399    * Output:
400    * Algo:
401    * Side Effects:
402    *************************************************************************/
403   void SelectPrintWin(wid, win)
404   Widget *wid;
405   Window win;
406   {
407     char str[40];
408     sprintf(str,"0x%x",win);
409
410     XtVaSetValues(wid[0],
411                   XmNvalue, str,
412                   NULL);
413   }
414
415   /*************************************************************************
416    * Input:
417    * Output:
418    * Algo:
419    * Side Effects:
420    *************************************************************************/
421   void SelectPrintName(wid, win)
422   Widget *wid;
423   Window win;
424   {
425    char *window_name;
426
427     XFetchName(EdInfo->ed_display,
428                win,&window_name);
429
430     XtVaSetValues(wid[1],
431                   XmNvalue, window_name,
432                   NULL);
433   }
434
435
436   /*************************************************************************
437    * Menu structures
438    *************************************************************************/
439   MenuItem file_menus[] = {
440       { "New", &xmPushButtonGadgetClass, 'N', NULL, NULL,
441           New_CB, 0,  (MenuItem *)NULL },
442       { "Save", &xmPushButtonGadgetClass, 'S', NULL, NULL,
443           Save_CB, 0,  (MenuItem *)NULL },
444       { "Retrieve", &xmPushButtonGadgetClass, 'N', NULL, NULL,
445           Retrieve_CB, 0,  (MenuItem *)NULL },
446       { "Quit", &xmPushButtonGadgetClass, 'Q', NULL, NULL,
447           Quit_CB, 0,  (MenuItem *)NULL },
448       NULL,
449   };
450
451   MenuItem state_menus[] = {
452       { "Normal", &xmPushButtonGadgetClass, 'N', NULL, NULL,
453           Normal_CB, 0,  (MenuItem *)NULL },
454       { "Record", &xmPushButtonGadgetClass, 'R', NULL, NULL,
455           Record_CB, 0,  (MenuItem *)NULL },
456       { "Play", &xmPushButtonGadgetClass, 'P', NULL, NULL,
457           Play_CB, 0,  (MenuItem *)NULL },
458       NULL,
459   };
460
461   MenuItem edit_menus[] = {
462       { "Compress On", &xmPushButtonGadgetClass, 'N', NULL, NULL,
463           0, 0, (MenuItem *)NULL },
464       { "Compress Off", &xmPushButtonGadgetClass, 'S', NULL, NULL,
465           0, 0, (MenuItem *)NULL },
466       NULL,
467   };
468
469
470   /*************************************************************************
471    * Input:
472    * Output:
473    * Algo:
474    * Side Effects:
475    *************************************************************************/
476   main(argc, argv)
477   int argc;
478   char *argv[];
479   {
480     Widget
481       toplevel,
482       main_w,
483       menubar,
484       form_w,
485       state_frame, state_form,
486       action_frame, action_form,
487       filter_form,
488       message_form;
489
490     XtAppContext app;
491
492     filter_mask[0] = NULL;
493     filter_mask[1] = NULL;
494
495     toplevel = XtVaAppInitialize(&app, "EvDaemon", NULL, 0,
496                                  &argc, argv, NULL, NULL);
497
498     /* Create a MainWindow widget that contains a DrawingArea in
499      * its work window. (This happens by default.)
500      */
501     main_w = XtVaCreateManagedWidget("main_w",
502                                      xmMainWindowWidgetClass, toplevel,
503                                      XmNscrollingPolicy,  XmAUTOMATIC,
504                                      XmNwidth, 500,
505                                      XmNheight, 345,
506                                      NULL);
507
508     menubar = XmCreateMenuBar(main_w, "menubar", NULL, 0);
509     BuildPulldownMenu(menubar, "File", 'F', file_menus);
510
511     XtManageChild(menubar);
512
```

```
513     form_w = XtVaCreateManagedWidget("form",
514              xmFormWidgetClass, main_w,
515              NULL);
516
517     state_frame = XtVaCreateManagedWidget("stateframe",
518                   xmFrameWidgetClass, form_w,
519                   XmNtopAttachment, XmATTACH_FORM,
520                   XmNtopOffset, 10,
521                   XmNleftAttachment, XmATTACH_FORM,
522                   XmNleftOffset, 10,
523                   XmNwidth, 200,
524                   XmNheight, 400,
525                   NULL);
526
527     state_form = XtVaCreateManagedWidget("s_form",
528                  xmFormWidgetClass, state_frame,
529                  XmNwidth, 70,
530                  XmNheight, 160,
531                  NULL);
532
533     SetupStatePanel(state_form);
534
535     action_frame = XtVaCreateManagedWidget("actionframe",
536                    xmFrameWidgetClass, form_w,
537                    XmNtopAttachment, XmATTACH_FORM,
538                    XmNtopOffset, 10,
539                    XmNleftAttachment, XmATTACH_WIDGET,
540                    XmNleftWidget, state_frame,
541                    XmNleftOffset, 10,
542                    NULL);
543
544     action_form = XtVaCreateManagedWidget("a_form",
545                   xmFormWidgetClass, action_frame,
546                   NULL);
547
548     SetupActionPanel(action_form, toplevel);
549
550     message_form = XtVaCreateManagedWidget("m_form",
551                    xmFormWidgetClass, form_w,
552                    XmNtopAttachment, XmATTACH_WIDGET,
553                    XmNtopWidget, action_frame,
554                    XmNtopOffset, 10,
555                    XmNleftAttachment, XmATTACH_FORM,
556                    XmNleftOffset, 10,
557                    XmNrightAttachment, XmATTACH_FORM,
558                    XmNrightOffset, 10,
559                    NULL);
560
561     SetupMessagePanel(message_form);
562
563     InitQueue(&Q);
564     XtRealizeWidget(toplevel);
565     EdInfo->ed_display = XtDisplay(toplevel);
566     EdInfo->ed_window = XtWindow(toplevel);
567     EdAppMainLoop(app);
568   }
569
```

```
/*****************************************************************************
 * uiAction.c
 *
 *****************************************************************************
#include <stdio.h>

#include <Xm/Label.h>
#include <Xm/RowColumn.h> /* Motif stuff */
#include <Xm/Form.h>
#include <Xm/ToggleBG.h>
#include <Xm/Text.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/Frame.h>

#include "ui.h"
#include "XemsEvent.h"

/*****************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 *****************************************************************************
void SetupActionPanel(form, top)
Widget form, top;
{
  Widget
    label,
    sel_frame, sel_form, sel_button, sel_win_id, sel_win_name,
    sel_win_id_label, sel_win_name_label,
    row_col, reset, send, pause, step, print, mesg_on, mesg_off,
    filter, fltr_popup;

  Position f_x, f_y;
  Dimension f_width, f_height;

  XmString str;
  Widget *sel_mesgs_ptr;

  sel_mesgs_ptr = (Widget*)XtMalloc(sizeof(Widget*)*2);

  str = XmStringCreateSimple("Action");
  label =  XtVaCreateManagedWidget("label",
           xmLabelWidgetClass, form,
           XmNtopAttachment, XmATTACH_FORM,
           XmNleftAttachment, XmATTACH_FORM,
           XmNrightAttachment, XmATTACH_FORM,
           XmNmarginLeft, 10,
           XmNmarginRight, 10,
           XmNmarginTop, 5,
           XmNmarginBottom, 10,
           XmNlabelString, str,
           NULL);
  XmStringFree(str);

  sel_frame = XtVaCreateManagedWidget("selframe",
                xmFrameWidgetClass, form,
                XmNtopAttachment, XmATTACH_WIDGET,
                XmNtopWidget, label,
                XmNleftAttachment, XmATTACH_FORM,
                XmNleftOffset, 5,
                XmNrightAttachment, XmATTACH_FORM,
                XmNrightOffset, 5,
                NULL);

  sel_form = XtVaCreateManagedWidget("selform",
               xmFormWidgetClass, sel_frame,
               NULL);

  str = XmStringCreateSimple("Window ID");
  sel_win_id_label = XtVaCreateManagedWidget("selwinidlab",
                     xmLabelWidgetClass, sel_form,
                     XmNlabelString, str,
                     XmNtopAttachment, XmATTACH_FORM,
                     XmNtopOffset, 10,
                     XmNleftAttachment, XmATTACH_FORM,
                     XmNalignment, XmALIGNMENT_END,
                     XmNmarginLeft, 10,
                     XmNmarginRight, 10,
                     XmNmarginTop, 5,
                     XmNmarginBottom, 10,
                     NULL);
  XmStringFree(str);

  sel_win_id = XtVaCreateManagedWidget("selwinid",
                 xmTextWidgetClass, sel_form,
                 XmNtopAttachment, XmATTACH_FORM,
                 XmNtopOffset, 10,
                 XmNleftAttachment, XmATTACH_WIDGET,
                 XmNleftWidget, sel_win_id_label,
                 XmNleftOffset, 5,
                 XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET,
                 XmNbottomWidget, sel_win_id_label,
                 XmNrows, 1,
                 XmNcolumns, 20,
                 XmNeditable, False,
                 XmNeditMode, XmSINGLE_LINE_EDIT,
                 NULL);

  str = XmStringCreateSimple("Name");
  sel_win_name_label = XtVaCreateManagedWidget("selwinnamelab",
                       xmLabelWidgetClass, sel_form,
                       XmNtopAttachment, XmATTACH_WIDGET,
                       XmNtopWidget, sel_win_id_label,
                       XmNleftAttachment, XmATTACH_FORM,
                       XmNbottomAttachment, XmATTACH_FORM,
                       XmNbottomOffset, 10,
                       XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET,
                       XmNrightWidget, sel_win_id_label,
                       XmNlabelString, str,
                       XmNalignment, XmALIGNMENT_END,
                       XmNmarginLeft, 10,
                       XmNmarginRight, 10,
                       XmNmarginTop, 5,
                       XmNmarginBottom, 10,
                       NULL);
  XmStringFree(str);

  sel_win_name = XtVaCreateManagedWidget("selwinname",
                   xmTextWidgetClass, sel_form,
                   XmNtopAttachment, XmATTACH_WIDGET,
                   XmNtopWidget, sel_win_id,
                   XmNtopOffset, 10,
                   XmNleftAttachment, XmATTACH_WIDGET,
                   XmNleftWidget, sel_win_name_label,
                   XmNleftOffset, 5,
                   XmNrightAttachment, XmATTACH_OPPOSITE_WIDGET,
```

```
129                     XmNrightWidget, sel_win_id,
130                     XmNbottomAttachment, XmATTACH_FORM,
131                     XmNbottomOffset, 10,
132                     XmNrows, 1,
133                     XmNcolumns, 20,
134                     XmNeditable, False,
135                     XmNeditMode, XmSINGLE_LINE_EDIT,
136                     NULL);
137
138
139    str = XmStringCreateSimple("Select Window");
140    sel_button = XtVaCreateManagedWidget("sel",
141             xmPushButtonWidgetClass, sel_form,
142             XmNtopAttachment, XmATTACH_FORM,
143             XmNtopOffset, 25,
144             XmNleftAttachment, XmATTACH_WIDGET,
145             XmNleftWidget, sel_win_id,
146             XmNleftOffset, 10,
147             XmNrightAttachment, XmATTACH_FORM,
148             XmNrightOffset, 10,
149             XmNwidth, 95,
150             XmNheight, 45,
151             XmNlabelString, str,
152             NULL);
153
154    sel_mesgs_ptr[0] = sel_win_id;
155    sel_mesgs_ptr[1] = sel_win_name;
156
157    XtAddCallback(sel_button, XmNactivateCallback,
158                  SelectWindow_CB, sel_mesgs_ptr);
159    XmStringFree(str);
160
161    row_col = XtVaCreateManagedWidget("row_col",
162              xmRowColumnWidgetClass, form,
163              XmNtopAttachment, XmATTACH_WIDGET,
164              XmNtopWidget, sel_frame,
165              XmNleftAttachment, XmATTACH_FORM,
166              XmNrightAttachment, XmATTACH_FORM,
167              XmNorientation, XmHORIZONTAL,
168              NULL);
169
170
171    reset = XtVaCreateManagedWidget("Reset",
172             xmPushButtonGadgetClass, row_col, NULL);
173    XtAddCallback(reset, XmNactivateCallback, QReset_CB, NULL);
174
175    send = XtVaCreateManagedWidget("Send",
176             xmPushButtonGadgetClass, row_col, NULL);
177    XtAddCallback(send, XmNactivateCallback, QSend_CB, 2);
178
179    pause   = XtVaCreateManagedWidget("Pause",
180             xmPushButtonGadgetClass, row_col, NULL);
181    XtAddCallback(pause, XmNactivateCallback, QStop_CB, 3);
182
183    step   = XtVaCreateManagedWidget("Step",
184             xmPushButtonGadgetClass, row_col, NULL);
185    XtAddCallback(step, XmNactivateCallback, QStep_CB, 3);
186
187    print   = XtVaCreateManagedWidget("Print",
188             xmPushButtonGadgetClass, row_col, NULL);
189    XtAddCallback(print, XmNactivateCallback, QPrint_CB, 3);
190
191    mesg_on   = XtVaCreateManagedWidget("MesgOn",
192             xmPushButtonGadgetClass, row_col, NULL);
193    XtAddCallback(mesg_on, XmNactivateCallback, MessageOn_CB, 3);
194
195    mesg_off   = XtVaCreateManagedWidget("MesgOff",
196             xmPushButtonGadgetClass, row_col, NULL);
197    XtAddCallback(mesg_off, XmNactivateCallback, MessageOff_CB, 3);
198
199    filter   = XtVaCreateManagedWidget("Filter",
200             xmPushButtonGadgetClass, row_col, NULL);
201
202    fltr_popup = XtVaCreatePopupShell("filterpopup",
203                                      transientShellWidgetClass,
204                                      form,
205                                      NULL);
206    XtVaSetValues(fltr_popup,
207                  XmNx, 0,
208                  XmNy, 0,
209                  NULL);
210
211    SetupFilterPanel(fltr_popup, top);
212
213    XtAddCallback(filter, XmNactivateCallback, FilterPop_CB, fltr_popup);
214
215    XtManageChild(row_col);
216  }
217
```

```
1   /*************************************************************************
2    * uiFilter.c
3    *
4    *************************************************************************/
5   #include <stdio.h>
6
7   #include <Xm/Label.h>
8   #include <Xm/RowColumn.h> /* Motif stuff */
9   #include <Xm/Form.h>
10  #include <Xm/ToggleBG.h>
11  #include <Xm/Text.h>
12  #include <Xm/PushB.h>
13  #include <Xm/PushBG.h>
14  #include <Xm/Frame.h>
15  #include "ui.h"
16
17  struct event_list
18  {
19    char *name;
20    int number;
21  } list[] =
22  {
23    "KeyPress", KeyPress,
24    "KeyRelease", KeyRelease,
25    "ButtonPress", ButtonPress,
26    "ButtonRelease", ButtonRelease,
27    "MotionNotify", MotionNotify,
28    "EnterNotify", EnterNotify,
29    "LeaveNotify", LeaveNotify,
30    "FocusIn", FocusIn,
31    "FocusOut", FocusOut,
32    "KeymapNotify", KeymapNotify,
33    "Expose", Expose,
34    "GraphicsExpose", GraphicsExpose,
35    "NoExpose", NoExpose,
36    "VisibilityNotify", VisibilityNotify,
37    "CreateNotify", CreateNotify,
38    "DestroyNotify", DestroyNotify,
39    "UnmapNotify", UnmapNotify,
40    "MapNotify", MapNotify,
41    "MapRequest", MapRequest,
42    "ReparentNotify", ReparentNotify,
43    "ConfigureNotify", ConfigureNotify,
44    "ConfigureRequest", ConfigureRequest,
45    "GravityNotify", GravityNotify,
46    "ResizeRequest", ResizeRequest,
47    "CirculateNotify", CirculateNotify,
48    "CirculateRequest", CirculateRequest,
49    "PropertyNotify", PropertyNotify,
50    "SelectionClear", SelectionClear,
51    "SelectionRequest", SelectionRequest,
52    "SelectionNotify", SelectionNotify,
53    "ColormapNotify", ColormapNotify,
54    "ClientMessage", ClientMessage,
55    "MappingNotify", MappingNotify
56    };
57
58  /*************************************************************************
59   * void SetupFilterPanel()
60   * Input:
61   * Output:
62   * Algo:
63   * Side Effects:
64   *************************************************************************/
65  void SetupFilterPanel(parent, toplevel)
66  Widget parent,toplevel;
67  {
68    Widget fpform, row_col, tog_but[36], down;
69    int i;
70
71    fpform =  XtVaCreateManagedWidget("fpform",
72                                xmFormWidgetClass, parent,
73                                NULL);
74
75    row_col = XtVaCreateWidget("row_col",
76                                xmRowColumnWidgetClass, fpform,
77                                XmNtopAttachment, XmATTACH_FORM,
78                                XmNleftAttachment, XmATTACH_FORM,
79                                XmNrightAttachment, XmATTACH_FORM,
80                                XmNorientation, XmVERTICAL,
81                                XmNnumColumns, 2,
82                                XmNpacking, XmPACK_COLUMN,
83                                NULL);
84
85    for (i=0;i<33;i++)
86      {
87        tog_but[i] = XtVaCreateManagedWidget(list[i].name,
88                      xmToggleButtonGadgetClass, row_col, NULL);
89        XtAddCallback(tog_but[i], XmNvalueChangedCallback,
90                       ButtonToggle_CB, list[i].number);
91      }
92
93
94    down = XtVaCreateManagedWidget("Done",
95                                   xmPushButtonGadgetClass, fpform,
96                                   XmNtopAttachment, XmATTACH_WIDGET,
97                                   XmNtopWidget, row_col,
98                                   XmNtopOffset, 10,
99                                   XmNleftAttachment, XmATTACH_FORM,
100                                  XmNleftOffset, 100,
101                                  XmNrightAttachment, XmATTACH_FORM,
102                                  XmNrightOffset, 100,
103                                  XmNbottomAttachment, XmATTACH_FORM,
104                                  XmNbottomOffset, 15,
105                                  NULL);
106
107   XtAddCallback(down, XmNactivateCallback, FilterDown_CB, parent);
108
109   XtManageChild(row_col);
110
111 }
```

```
1  /*****************************************************************************
2   * uiMenu.c
3   *
4   *****************************************************************************
5
6  #include <Xm/ToggleBG.h>
7  #include <Xm/PushBG.h>
8  #include <Xm/CascadeBG.h>
9  #include "ui.h"
10
11
12
13 /*****************************************************************************
14  * Widget BuildPulldownMenu(parent, menu_title, menu_mnemonic, items)
15  *
16  * Pulldown menus are built from cascade buttons, so this function
17  * also includes pullright menus.  Create the menu, the cascade button
18  * that owns the menu, and then the submenu items.
19  *****************************************************************************/
20 Widget BuildPulldownMenu(parent, menu_title, menu_mnemonic, items)
21 Widget parent;
22 char *menu_title, menu_mnemonic;
23 MenuItem *items;
24 {
25     Widget PullDown, cascade, widget;
26     int i;
27     XmString str;
28
29     PullDown = (Widget)XmCreatePulldownMenu(parent, "_pulldown", NULL, 0);
30
31     str = XmStringCreateSimple(menu_title);
32     cascade = XtVaCreateManagedWidget(menu_title,
33         xmCascadeButtonGadgetClass, parent,
34         XmNsubMenuId,   PullDown,
35         XmNlabelString, str,
36         XmNmnemonic,    menu_mnemonic,
37         NULL);
38     XmStringFree(str);
39
40     /* Now add the menu items */
41     for (i = 0; items[i].label != NULL; i++) {
42         /* If subitems exist, create the pull-right menu by calling this
43          * function recursively.  Since the function returns a cascade
44          * button, the widget returned is used..
45          */
46         if (items[i].subitems)
47             widget = BuildPulldownMenu(PullDown,
48                 items[i].label, items[i].mnemonic, items[i].subitems);
49         else
50             widget = XtVaCreateManagedWidget(items[i].label,
51                 *items[i].class, PullDown,
52                 NULL);
53         /* Whether the item is a real item or a cascade button with a
54          * menu, it can still have a mnemonic.
55          */
56         if (items[i].mnemonic)
57             XtVaSetValues(widget, XmNmnemonic, items[i].mnemonic, NULL);
58         /* any item can have an accelerator, except cascade menus. But,
59          * we don't worry about that; we know better in our declarations.
60          */
61         if (items[i].accelerator) {
62             str = XmStringCreateSimple(items[i].accel_text);
63             XtVaSetValues(widget,
64                 XmNaccelerator, items[i].accelerator,
65                 XmNacceleratorText, str,
66                 NULL);
67             XmStringFree(str);
68         }
69         /* again, anyone can have a callback -- however, this is an
70          * activate-callback.  This may not be appropriate for all items.
71          */
72         if (items[i].callback)
73             XtAddCallback(widget, XmNactivateCallback,
74                 items[i].callback, items[i].callback_data);
75     }
76     return cascade;
77 }
```

```
1  /*************************************************************************
2   * uiMessage.c
3   *
4   *************************************************************************/
5  #include <stdio.h>
6  #include <string.h>
7  #include <Xm/Text.h>
8  #include <Xm/ScrolledW.h>
9  #include <Xm/Label.h>
10
11 Widget msg, label;
12
13
14 /************************************************************************
15  * Input:
16  * Output:
17  * Algo:
18  * Side Effects:
19  ************************************************************************/
20 void SetupMessagePanel(form)
21 Widget form;
22 {
23   XmString str;
24
25   str = XmStringCreateSimple("Messages");
26   label =  XtVaCreateManagedWidget("label",
27            xmLabelWidgetClass, form,
28            XmNtopAttachment, XmATTACH_FORM,
29            XmNleftAttachment, XmATTACH_FORM,
30            XmNrightAttachment, XmATTACH_FORM,
31            XmNmarginLeft, 10,
32            XmNmarginRight, 10,
33            XmNmarginTop, 5,
34            XmNmarginBottom, 10,
35            XmNlabelString, str,
36            NULL);
37   XmStringFree(str);
38
39
40
41   msg = XmCreateScrolledText(form,"msg",NULL, 0);
42   XtVaSetValues(XtParent(msg),
43                 XmNtopAttachment, XmATTACH_WIDGET,
44                 XmNtopWidget, label,
45                 NULL);
46
47   XtVaSetValues(msg,
48                 XmNrows, 4,
49                 XmNcolumns, 70,
50                 XmNeditable, False,
51                 XmNeditMode, XmMULTI_LINE_EDIT,
52                 XmNcursorPositionVisible, False,
53                 XmNsensitive, False,
54                 XmNscrollHorizontal, False,
55                 XmNwordWrap, True,
56                 NULL);
57
58   XtManageChild(msg);
59 }
60
61
62
63 /************************************************************************
64  * Input:
65  * Output:
66  * Algo:
67  * Side Effects:
68  ************************************************************************/
69 static XmTextPosition position;
70
71 void MessagePrint(messg)
72 char *messg;
73 {
74   XmTextInsert(msg, position, messg);
75   position += strlen(messg);
76   XtVaSetValues(msg, XmNcursorPosition, position, NULL);
77   XmTextShowPosition(msg, position);
78 }
```

```c
/*****************************************************************************
 * uiState.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h> /* Motif stuff */
#include <Xm/Form.h>
#include <Xm/ToggleBG.h>
#include "ui.h"

/*************************************************************************
 * Input:
 * Output:
 * Algo:
 * Side Effects:
 *************************************************************************/
void SetupStatePanel(form)
Widget form;
{
  Widget label, radio_box, normal, record, play;


  XmString str = XmStringCreateSimple("State");

  label =  XtVaCreateManagedWidget("label",
           xmLabelWidgetClass, form,
           XmNtopAttachment, XmATTACH_FORM,
           XmNleftAttachment, XmATTACH_FORM,
           XmNrightAttachment, XmATTACH_FORM,
           XmNmarginLeft, 10,
           XmNmarginRight, 10,
           XmNmarginTop, 5,
           XmNmarginBottom, 10,
           XmNlabelString, str,
           NULL);

  radio_box = XmCreateRadioBox(form, "radio_box", NULL, 0);
  XtVaSetValues(radio_box,
                XmNtopAttachment, XmATTACH_WIDGET,
                XmNtopWidget, label,
                XmNtopOffset, 15,
                XmNleftAttachment, XmATTACH_FORM,
                NULL);


  normal = XtVaCreateManagedWidget("Normal",
           xmToggleButtonGadgetClass, radio_box, NULL);
  XtAddCallback(normal, XmNvalueChangedCallback, Normal_CB, 1);

  record = XtVaCreateManagedWidget("Record",
           xmToggleButtonGadgetClass, radio_box, NULL);
  XtAddCallback(record, XmNvalueChangedCallback, Record_CB, 2);

  play   = XtVaCreateManagedWidget("Play",
           xmToggleButtonGadgetClass, radio_box, NULL);
  XtAddCallback(play, XmNvalueChangedCallback, Play_CB, 3);

  XtManageChild(radio_box);
}
```

```
1  ***********************************************************************
2   util.h
3   ***********************************************************************
4
5  long GetCurrentTime();
6  void PrintEvent();
```

```
/*************************************************************************
 util.c
 *************************************************************************/
#include <stdio.h>
#include <sys/time.h>
#include <X11/Xos.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>

#include "util.h"

long GetCurrentTime()
{
  struct timeval tp;
  struct timezone tpz;
  long time_msec;

  gettimeofday(&tp,&tpz);
  time_msec = tp.tv_sec * 1000 + tp.tv_usec/1000;

  return(time_msec);
}

/*************************************************************************
 void PrintEvent();
 *************************************************************************/
/*
 * xev - event diagnostics
 *
 * $XConsortium: xev.c,v 1.14 91/05/04 23:15:11 keith Exp $
 *
 * Copyright 1988 Massachusetts Institute of Technology
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appear in all copies and that both that
 * copyright notice and this permission notice appear in supporting
 * documentation, and that the name of M.I.T. not be used in advertising or
 * publicity pertaining to distribution of the software without specific,
 * written prior permission.  M.I.T. makes no representations about the
 * suitability of this software for any purpose.  It is provided "as is"
 * without express or implied warranty.
 *
 * Author:  Jim Fulton, MIT X Consortium
 */

char *Yes = "YES";
char *No = "NO";
char *Unknown = "unknown";

void PrintEvent(XEvent *event)
{
  switch (event->type)
    {
    case KeyPress:
      prologue (event, "KeyPress");
      do_KeyPress (event);
      break;
    case KeyRelease:
      prologue (event, "KeyRelease");
      do_KeyRelease (event);
    break;
    case ButtonPress:
      prologue (event, "ButtonPress");
      do_ButtonPress (event);
      break;
    case ButtonRelease:
      prologue (event, "ButtonRelease");
      do_ButtonRelease (event);
      break;
    case MotionNotify:
      prologue (event, "MotionNotify");
      do_MotionNotify (event);
      break;
    case EnterNotify:
      prologue (event, "EnterNotify");
      do_EnterNotify (event);
      break;
    case LeaveNotify:
      prologue (event, "LeaveNotify");
      do_LeaveNotify (event);
      break;
    case FocusIn:
      prologue (event, "FocusIn");
      do_FocusIn (event);
      break;
    case FocusOut:
      prologue (event, "FocusOut");
      do_FocusOut (event);
      break;
    case KeymapNotify:
      prologue (event, "KeymapNotify");
      do_KeymapNotify (event);
      break;
    case Expose:
      prologue (event, "Expose");
      do_Expose (event);
      break;
    case GraphicsExpose:
      prologue (event, "GraphicsExpose");
      do_GraphicsExpose (event);
      break;
    case NoExpose:
      prologue (event, "NoExpose");
      do_NoExpose (event);
      break;
    case VisibilityNotify:
      prologue (event, "VisibilityNotify");
      do_VisibilityNotify (event);
      break;
    case CreateNotify:
      prologue (event, "CreateNotify");
      do_CreateNotify (event);
      break;
    case DestroyNotify:
      prologue (event, "DestroyNotify");
      do_DestroyNotify (event);
      break;
    case UnmapNotify:
      prologue (event, "UnmapNotify");
      do_UnmapNotify (event);
      break;
    case MapNotify:
      prologue (event, "MapNotify");
      do_MapNotify (event);
      break;
    case MapRequest:
      prologue (event, "MapRequest");
```

```
129         do_MapRequest (event);
130         break;
131       case ReparentNotify:
132         prologue (event, "ReparentNotify");
133         do_ReparentNotify (event);
134         break;
135       case ConfigureNotify:
136         prologue (event, "ConfigureNotify");
137         do_ConfigureNotify (event);
138         break;
139       case ConfigureRequest:
140         prologue (event, "ConfigureRequest");
141         do_ConfigureRequest (event);
142         break;
143       case GravityNotify:
144         prologue (event, "GravityNotify");
145         do_GravityNotify (event);
146         break;
147       case ResizeRequest:
148         prologue (event, "ResizeRequest");
149         do_ResizeRequest (event);
150         break;
151       case CirculateNotify:
152         prologue (event, "CirculateNotify");
153         do_CirculateNotify (event);
154         break;
155       case CirculateRequest:
156         prologue (event, "CirculateRequest");
157         do_CirculateRequest (event);
158         break;
159       case PropertyNotify:
160         prologue (event, "PropertyNotify");
161         do_PropertyNotify (event);
162         break;
163       case SelectionClear:
164         prologue (event, "SelectionClear");
165         do_SelectionClear (event);
166         break;
167       case SelectionRequest:
168         prologue (event, "SelectionRequest");
169         do_SelectionRequest (event);
170         break;
171       case SelectionNotify:
172         prologue (event, "SelectionNotify");
173         do_SelectionNotify (event);
174         break;
175       case ColormapNotify:
176         prologue (event, "ColormapNotify");
177         do_ColormapNotify (event);
178         break;
179       case ClientMessage:
180         prologue (event, "ClientMessage");
181         do_ClientMessage (event);
182         break;
183       case MappingNotify:
184         prologue (event, "MappingNotify");
185         do_MappingNotify (event);
186         break;
187       default:
188         printf ("Unknown event type %d\n", event->type);
189         break;
190       }
191   }
192
```

```
193   prologue (eventp, event_name)
194       XEvent *eventp;
195       char *event_name;
196   {
197       XAnyEvent *e = (XAnyEvent *) eventp;
198
199       printf ("\n%s event, serial %ld, synthetic %s, window 0x%lx,\n",
200               event_name, e->serial, e->send_event ? Yes : No, e->window);
201       printf("Display: 0x%lx\n",e->display);
202
203   /*
204       printf("Widget Name:%s\n",XtName(XtWindowToWidget(e->display,e->window)));
205   */
206
207       return;
208   }
209
210
211   do_KeyPress (eventp)
212       XEvent *eventp;
213   {
214       XKeyEvent *e = (XKeyEvent *) eventp;
215       KeySym ks;
216       char *ksname;
217       int nbytes;
218       char str[256+1];
219
220       nbytes = XLookupString (e, str, 256, &ks, NULL);
221       if (ks == NoSymbol)
222           ksname = "NoSymbol";
223       else if (!(ksname = XKeysymToString (ks)))
224           ksname = "(no name)";
225       printf ("    root 0x%lx, subw 0x%lx, time %lu, (%d,%d), root:(%d,%d),\n",
226               e->root, e->subwindow, e->time, e->x, e->y, e->x_root, e->y_root);
227       printf ("    state 0x%x, keycode %u (keysym 0x%x, %s), same_screen %s,\n",
228               e->state, e->keycode, ks, ksname, e->same_screen ? Yes : No);
229       if (nbytes < 0) nbytes = 0;
230       if (nbytes > 256) nbytes = 256;
231       str[nbytes] = '\0';
232       printf ("    XLookupString gives %d characters:  \"%s\"\n", nbytes, str);
233
234       return;
235   }
236
237   do_KeyRelease (eventp)
238       XEvent *eventp;
239   {
240       do_KeyPress (eventp);                /* since it has the same info */
241       return;
242   }
243
244   do_ButtonPress (eventp)
245       XEvent *eventp;
246   {
247       XButtonEvent *e = (XButtonEvent *) eventp;
248
249       printf ("    root 0x%lx, subw 0x%lx, time %lu, (%d,%d), root:(%d,%d),\n",
250               e->root, e->subwindow, e->time, e->x, e->y, e->x_root, e->y_root);
251       printf ("    state 0x%x, button %u, same_screen %s\n",
252               e->state, e->button, e->same_screen ? Yes : No);
253
254       return;
255   }
256
```

```
do_ButtonRelease (eventp)
    XEvent *eventp;
{
    do_ButtonPress (eventp);            /* since it has the same info */
    return;
}

do_MotionNotify (eventp)
    XEvent *eventp;
{
    XMotionEvent *e = (XMotionEvent *) eventp;

    printf ("    root 0x%lx, subw 0x%lx, time %lu, (%d,%d), root:(%d,%d),\n",
            e->root, e->subwindow, e->time, e->x, e->y, e->x_root, e->y_root);
    printf ("    state 0x%x, is_hint %u, same_screen %s\n",
            e->state, e->is_hint, e->same_screen ? Yes : No);

    return;
}

do_EnterNotify (eventp)
    XEvent *eventp;
{
    XCrossingEvent *e = (XCrossingEvent *) eventp;
    char *mode, *detail;
    char dmode[10], ddetail[10];

    switch (e->mode) {
      case NotifyNormal:  mode = "NotifyNormal"; break;
      case NotifyGrab:  mode = "NotifyGrab"; break;
      case NotifyUngrab:  mode = "NotifyUngrab"; break;
      case NotifyWhileGrabbed:  mode = "NotifyWhileGrabbed"; break;
      default:  mode = dmode, sprintf (dmode, "%u", e->mode); break;
    }

    switch (e->detail) {
      case NotifyAncestor:  detail = "NotifyAncestor"; break;
      case NotifyVirtual:  detail = "NotifyVirtual"; break;
      case NotifyInferior:  detail = "NotifyInferior"; break;
      case NotifyNonlinear:  detail = "NotifyNonlinear"; break;
      case NotifyNonlinearVirtual:  detail = "NotifyNonlinearVirtual"; break;
      case NotifyPointer:  detail = "NotifyPointer"; break;
      case NotifyPointerRoot:  detail = "NotifyPointerRoot"; break;
      case NotifyDetailNone:  detail = "NotifyDetailNone"; break;
      default:  detail = ddetail; sprintf (ddetail, "%u", e->detail); break;
    }

    printf ("    root 0x%lx, subw 0x%lx, time %lu, (%d,%d), root:(%d,%d),\n",
            e->root, e->subwindow, e->time, e->x, e->y, e->x_root, e->y_root);
    printf ("    mode %s, detail %s, same_screen %s,\n",
            mode, detail, e->same_screen ? Yes : No);
    printf ("    focus %s, state %u\n", e->focus ? Yes : No, e->state);

    return;
}

do_LeaveNotify (eventp)
    XEvent *eventp;
{
    do_EnterNotify (eventp);            /* since it has same information */
    return;
}

do_FocusIn (eventp)
    XEvent *eventp;
{
    XFocusChangeEvent *e = (XFocusChangeEvent *) eventp;
    char *mode, *detail;
    char dmode[10], ddetail[10];

    switch (e->mode) {
      case NotifyNormal:  mode = "NotifyNormal"; break;
      case NotifyGrab:  mode = "NotifyGrab"; break;
      case NotifyUngrab:  mode = "NotifyUngrab"; break;
      case NotifyWhileGrabbed:  mode = "NotifyWhileGrabbed"; break;
      default:  mode = dmode, sprintf (dmode, "%u", e->mode); break;
    }

    switch (e->detail) {
      case NotifyAncestor:  detail = "NotifyAncestor"; break;
      case NotifyVirtual:  detail = "NotifyVirtual"; break;
      case NotifyInferior:  detail = "NotifyInferior"; break;
      case NotifyNonlinear:  detail = "NotifyNonlinear"; break;
      case NotifyNonlinearVirtual:  detail = "NotifyNonlinearVirtual"; break;
      case NotifyPointer:  detail = "NotifyPointer"; break;
      case NotifyPointerRoot:  detail = "NotifyPointerRoot"; break;
      case NotifyDetailNone:  detail = "NotifyDetailNone"; break;
      default:  detail = ddetail; sprintf (ddetail, "%u", e->detail); break;
    }

    printf ("    mode %s, detail %s\n", mode, detail);
    return;
}

do_FocusOut (eventp)
    XEvent *eventp;
{
    do_FocusIn (eventp);                /* since it has same information */
    return;
}

do_KeymapNotify (eventp)
    XEvent *eventp;
{
    XKeymapEvent *e = (XKeymapEvent *) eventp;
    int i;

    printf ("    keys:  ");
    for (i = 0; i < 32; i++) {
        if (i == 16) printf ("\n            ");
        printf ("%-3u ", (unsigned int) e->key_vector[i]);
    }
    printf ("\n");
    return;
}

do_Expose (eventp)
    XEvent *eventp;
{
    XExposeEvent *e = (XExposeEvent *) eventp;

    printf ("    (%d,%d), width %d, height %d, count %d\n",
            e->x, e->y, e->width, e->height, e->count);
    return;
}

do_GraphicsExpose (eventp)
    XEvent *eventp;
```

```
385  {
386      XGraphicsExposeEvent *e = (XGraphicsExposeEvent *) eventp;
387      char *m;
388      char mdummy[10];
389
390      switch (e->major_code) {
391        default:  m = mdummy; sprintf (mdummy, "%d", e->major_code); break;
392      }
393
394      printf ("    (%d,%d), width %d, height %d, count %d,\n",
395              e->x, e->y, e->width, e->height, e->count);
396      printf ("    major %s, minor %d\n", m, e->minor_code);
397      return;
398  }
399
400  do_NoExpose (eventp)
401      XEvent *eventp;
402  {
403      XNoExposeEvent *e = (XNoExposeEvent *) eventp;
404      char *m;
405      char mdummy[10];
406
407      switch (e->major_code) {
408        default:  m = mdummy; sprintf (mdummy, "%d", e->major_code); break;
409      }
410
411      printf ("    major %s, minor %d\n", m, e->minor_code);
412      return;
413  }
414
415  do_VisibilityNotify (eventp)
416      XEvent *eventp;
417  {
418      XVisibilityEvent *e = (XVisibilityEvent *) eventp;
419      char *v;
420      char vdummy[10];
421
422      switch (e->state) {
423        case VisibilityUnobscured:  v = "VisibilityUnobscured"; break;
424        case VisibilityPartiallyObscured:  v = "VisibilityPartiallyObscured"; bre
ak;
425        case VisibilityFullyObscured:  v = "VisibilityFullyObscured"; break;
426        default:  v = vdummy; sprintf (vdummy, "%d", e->state); break;
427      }
428
429      printf ("    state %s\n", v);
430      return;
431  }
432
433  do_CreateNotify (eventp)
434      XEvent *eventp;
435  {
436      XCreateWindowEvent *e = (XCreateWindowEvent *) eventp;
437
438      printf ("    parent 0x%lx, window 0x%lx, (%d,%d), width %d, height %d,\n",
439              e->parent, e->window, e->x, e->y, e->width, e->height);
440      printf ("border_width %d, override %s\n",
441              e->border_width, e->override_redirect ? Yes : No);
442      return;
443  }
444
445  do_DestroyNotify (eventp)
446      XEvent *eventp;
447  {
448      XDestroyWindowEvent *e = (XDestroyWindowEvent *) eventp;
449
450      printf ("    event 0x%lx, window 0x%lx\n", e->event, e->window);
451      return;
452  }
453
454  do_UnmapNotify (eventp)
455      XEvent *eventp;
456  {
457      XUnmapEvent *e = (XUnmapEvent *) eventp;
458
459      printf ("    event 0x%lx, window 0x%lx, from_configure %s\n",
460              e->event, e->window, e->from_configure ? Yes : No);
461      return;
462  }
463
464  do_MapNotify (eventp)
465      XEvent *eventp;
466  {
467      XMapEvent *e = (XMapEvent *) eventp;
468
469      printf ("    event 0x%lx, window 0x%lx, override %s\n",
470              e->event, e->window, e->override_redirect ? Yes : No);
471      return;
472  }
473
474  do_MapRequest (eventp)
475      XEvent *eventp;
476  {
477      XMapRequestEvent *e = (XMapRequestEvent *) eventp;
478
479      printf ("    parent 0x%lx, window 0x%lx\n", e->parent, e->window);
480      return;
481  }
482
483  do_ReparentNotify (eventp)
484      XEvent *eventp;
485  {
486      XReparentEvent *e = (XReparentEvent *) eventp;
487
488      printf ("    event 0x%lx, window 0x%lx, parent 0x%lx,\n",
489              e->event, e->window, e->parent);
490      printf ("    (%d,%d), override %s\n", e->x, e->y,
491              e->override_redirect ? Yes : No);
492      return;
493  }
494
495  do_ConfigureNotify (eventp)
496      XEvent *eventp;
497  {
498      XConfigureEvent *e = (XConfigureEvent *) eventp;
499
500      printf ("    event 0x%lx, window 0x%lx, (%d,%d), width %d, height %d,\n",
501              e->event, e->window, e->x, e->y, e->width, e->height);
502      printf ("    border_width %d, above 0x%lx, override %s\n",
503              e->border_width, e->above, e->override_redirect ? Yes : No);
504      return;
505  }
506
507  do_ConfigureRequest (eventp)
508      XEvent *eventp;
509  {
510      XConfigureRequestEvent *e = (XConfigureRequestEvent *) eventp;
511      char *detail;
```

```
    char ddummy[10];

    switch (e->detail) {
      case Above:  detail = "Above";  break;
      case Below:  detail = "Below";  break;
      case TopIf:  detail = "TopIf";  break;
      case BottomIf:  detail = "BottomIf"; break;
      case Opposite:  detail = "Opposite"; break;
      default:  detail = ddummy; sprintf (ddummy, "%d", e->detail); break;
    }

    printf ("    parent 0x%lx, window 0x%lx, (%d,%d), width %d, height %d,\n",
            e->parent, e->window, e->x, e->y, e->width, e->height);
    printf ("    border_width %d, above 0x%lx, detail %s, value 0x%lx\n",
            e->border_width, e->above, detail, e->value_mask);
    return;
}

do_GravityNotify (eventp)
    XEvent *eventp;
{
    XGravityEvent *e = (XGravityEvent *) eventp;

    printf ("    event 0x%lx, window 0x%lx, (%d,%d)\n",
            e->event, e->window, e->x, e->y);
    return;
}

do_ResizeRequest (eventp)
    XEvent *eventp;
{
    XResizeRequestEvent *e = (XResizeRequestEvent *) eventp;

    printf ("    width %d, height %d\n", e->width, e->height);
    return;
}

do_CirculateNotify (eventp)
    XEvent *eventp;
{
    XCirculateEvent *e = (XCirculateEvent *) eventp;
    char *p;
    char pdummy[10];

    switch (e->place) {
      case PlaceOnTop:  p = "PlaceOnTop"; break;
      case PlaceOnBottom:  p = "PlaceOnBottom"; break;
      default:  p = pdummy; sprintf (pdummy, "%d", e->place); break;
    }

    printf ("    event 0x%lx, window 0x%lx, place %s\n",
            e->event, e->window, p);
    return;
}

do_CirculateRequest (eventp)
    XEvent *eventp;
{
    XCirculateRequestEvent *e = (XCirculateRequestEvent *) eventp;
    char *p;
    char pdummy[10];

    switch (e->place) {
      case PlaceOnTop:  p = "PlaceOnTop"; break;
      case PlaceOnBottom:  p = "PlaceOnBottom"; break;
      default:  p = pdummy; sprintf (pdummy, "%d", e->place); break;
    }

    printf ("    parent 0x%lx, window 0x%lx, place %s\n",
            e->parent, e->window, p);
    return;
}


do_PropertyNotify (eventp)
    XEvent *eventp;
{
    XPropertyEvent *e = (XPropertyEvent *) eventp;
    char *aname = XGetAtomName (eventp->xany.display, e->atom);
    char *s;
    char sdummy[10];

    switch (e->state) {
      case PropertyNewValue:  s = "PropertyNewValue"; break;
      case PropertyDelete:  s = "PropertyDelete"; break;
      default:  s = sdummy; sprintf (sdummy, "%d", e->state); break;
    }

    printf ("    atom 0x%lx (%s), time %lu, state %s\n",
           e->atom, aname ? aname : Unknown, e->time,  s);

    if (aname) XFree (aname);
    return;
}

do_SelectionClear (eventp)
    XEvent *eventp;
{
    XSelectionClearEvent *e = (XSelectionClearEvent *) eventp;
    char *sname = XGetAtomName (eventp->xany.display, e->selection);

    printf ("    selection 0x%lx (%s), time %lu\n",
            e->selection, sname ? sname : Unknown, e->time);

    if (sname) XFree (sname);
    return;
}

do_SelectionRequest (eventp)
    XEvent *eventp;
{
    XSelectionRequestEvent *e = (XSelectionRequestEvent *) eventp;
    char *sname = XGetAtomName (eventp->xany.display, e->selection);
    char *tname = XGetAtomName (eventp->xany.display, e->target);
    char *pname = XGetAtomName (eventp->xany.display, e->property);

    printf ("    owner 0x%lx, requestor 0x%lx, selection 0x%lx (%s),\n",
            e->owner, e->requestor, e->selection, sname ? sname : Unknown);
    printf ("    target 0x%lx (%s), property 0x%lx (%s), time %lu\n",
            e->target, tname ? tname : Unknown, e->property,
            pname ? pname : Unknown, e->time);

    if (sname) XFree (sname);
    if (tname) XFree (tname);
    if (pname) XFree (pname);

    return;
}
```

```
640
641 do_SelectionNotify (eventp)
642     XEvent *eventp;
643 {
644     XSelectionEvent *e = (XSelectionEvent *) eventp;
645     char *sname = XGetAtomName (eventp->xany.display, e->selection);
646     char *tname = XGetAtomName (eventp->xany.display, e->target);
647     char *pname = XGetAtomName (eventp->xany.display, e->property);
648
649     printf ("    selection 0x%lx (%s), target 0x%lx (%s),\n",
650             e->selection, sname ? sname : Unknown, e->target,
651             tname ? tname : Unknown);
652     printf ("    property 0x%lx (%s), time %lu\n",
653             e->property, pname ? pname : Unknown, e->time);
654
655     if (sname) XFree (sname);
656     if (tname) XFree (tname);
657     if (pname) XFree (pname);
658
659     return;
660 }
661
662 do_ColormapNotify (eventp)
663     XEvent *eventp;
664 {
665     XColormapEvent *e = (XColormapEvent *) eventp;
666     char *s;
667     char sdummy[10];
668
669     switch (e->state) {
670       case ColormapInstalled:  s = "ColormapInstalled"; break;
671       case ColormapUninstalled:  s = "ColormapUninstalled"; break;
672       default:  s = sdummy; sprintf (sdummy, "%d", e->state); break;
673     }
674
675     printf ("    colormap 0x%lx, new %s, state %s\n",
676             e->colormap, e->new ? Yes : No, s);
677     return;
678 }
679
680 /*
681 do_ClientMessage (eventp)
682     XEvent *eventp;
683 {
684     XClientMessageEvent *e = (XClientMessageEvent *) eventp;
685     char *mname = XGetAtomName (eventp->xany.display, e->message_type);
686
687     printf ("    message_type 0x%lx (%s), format %d\n",
688             e->message_type, mname ? mname : Unknown, e->format);
689
690     if (mname) XFree (mname);
691     return;
692 }
693 */
694
695
696 do_ClientMessage (eventp)
697     XEvent *eventp;
698 {
699   int i;
700
701   XClientMessageEvent *e = (XClientMessageEvent *) eventp;
702   printf ("Client Message\n");
703
704   printf ("    event 0x%lx, window 0x%lx, message type %d\n",
705           e->type, e->window, e->message_type);
706
707   printf("Data:\n");
708     printf("%s",e->data.b);
709   printf("\n");
710   return;
711 }
712
713
714
715
716 do_MappingNotify (eventp)
717     XEvent *eventp;
718 {
719     XMappingEvent *e = (XMappingEvent *) eventp;
720     char *r;
721     char rdummy[10];
722
723     switch (e->request) {
724       case MappingModifier:  r = "MappingModifier"; break;
725       case MappingKeyboard:  r = "MappingKeyboard"; break;
726       case MappingPointer:  r = "MappingPointer"; break;
727       default:  r = rdummy; sprintf (rdummy, "%d", e->request); break;
728     }
729
730     printf ("    request %s, first_keycode %d, count %d\n",
731             r, e->first_keycode, e->count);
732     XRefreshKeyboardMapping(e);
733     return;
734 }
735
736
```