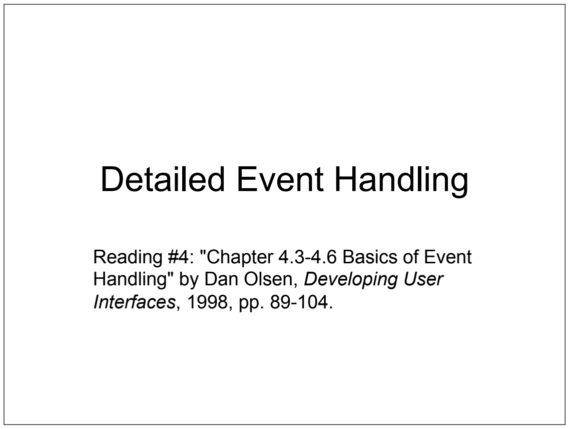
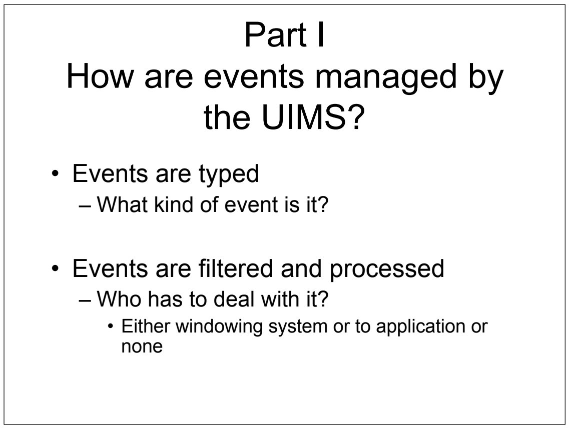


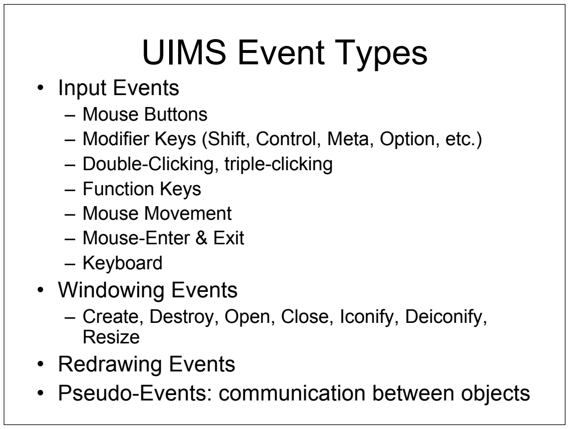
Detailed Event Handling

Reading #4: "Chapter 4.3-4.6 Basics of Event Handling" by Dan Olsen, *Developing User Interfaces*, 1998, pp. 89-104.



Part I
How are events managed by the UIMS?

- Events are typed
  - What kind of event is it?
- Events are filtered and processed
  - Who has to deal with it?
    - Either windowing system or to application or none



UIMS Event Types

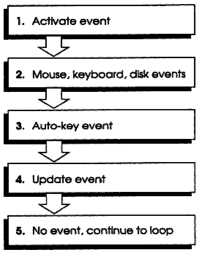
- Input Events
  - Mouse Buttons
  - Modifier Keys (Shift, Control, Meta, Option, etc.)
  - Double-Clicking, triple-clicking
  - Function Keys
  - Mouse Movement
  - Mouse-Enter & Exit
  - Keyboard
- Windowing Events
  - Create, Destroy, Open, Close, Iconify, Deiconify, Resize
- Redrawing Events
- Pseudo-Events: communication between objects

How are events managed by the UIMS?

cont.

- Events are filtered
  - Either windowing system or to application or none
- Event priority queue managed by OS
- Ordered by
  - Priorities pre-set by OS for event types
  - Timestamp
- Macintosh and Microsoft Windows have only one queue
- Multi-tasking OS (e.g. X Window) has a queue for each process

Macintosh Event Priority Queue



1. Activate event
2. Mouse, keyboard, disk events
3. Auto-key event
4. Update event
5. No event, continue to loop

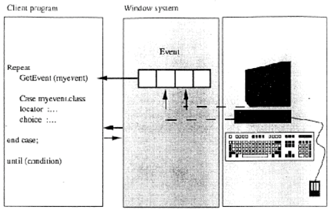
- Activate event: activate specific window
- Disk event: Insert diskette
- Auto-key event: repeated key
- Update event: redraw window

How are events managed by the UIMS?

cont.

- Events are records sent by the windowing system to the application
  - name of event
  - timestamp
  - event-specific fields such as XY location for pointing device
  - widget object or window ID

UIMS Event Processing



```
Repeat
    GetEvent (myevent)

    Case myevent.class
    locator :...
    choice  :...

end case;

until (condition)
```

FIGURE 3.1
Principles of input operation of an abstract terminal. The client program is structured to loop through receiving and processing events continually. The window system translates the operator's actions into events for the client program.

Event Record

```
Event = Record
    EventCode: Integer;
    MouseX, MouseY:Integer;
    EventValue: Integer;
    Time:Integer;
    WindowID:Integer;
End;
```

where EventCode “1” for mouse button;
EventValue “2” for down

How are events managed by the UIMS?

cont.

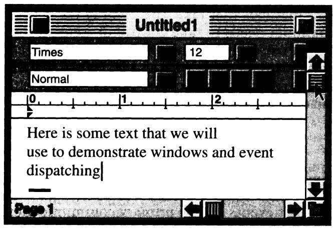
How does the windowing system associate the event with a window?

Called “event dispatching”

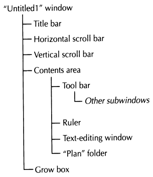
- Hierarchy of windows
  - bottom-first processing
- Input focus
  - Currently selected window receives all key & mouse events

Event Dispatching

Hierarchy of windows



Here is some text that we will use to demonstrate windows and event dispatching



- "Untitled1" window
  - Title bar
  - Horizontal scroll bar
  - Vertical scroll bar
  - Contents area
    - Tool bar
      - *Other subwindows*
    - Ruler
    - Text-editing window
    - "Plan" folder
  - Grow box

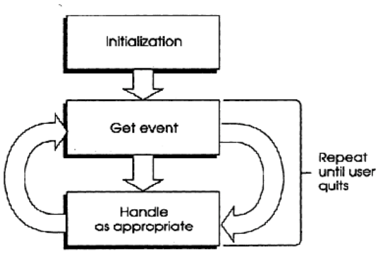
Part II
Event management within the program

- Main Event loop
  - Procedural languages
    - Explicit main event loop
    - Procedure name, event table, callbacks
  - Object-oriented languages
    - Implicit main event loop
    - Event handlers

Explicit Event Handling in the Application Program

- Trap calls to ROM-based Toolbox code
  - Example: Macintosh Pascal would use "case" statement
- Event-table
  - Each window has a pointer to an event table for each possible event
  - Event table has addresses for procedures to handle various event types
  - Example: Applications written completely in C

The Main Event Loop



Explicit Main Event Loop

Mac Toolb Event!

```
/*
 * Loop for ever on event acquisition
 */
while (go) {

/*
 * Get next event of any type and process it according to its type
 */
            if (GetNextEvent(everyEvent, myevent))
                  switch (myevent->what) {
                        case keyDown : ..... break;
                        case mouseUp :
                        case mouseDown:
/*
 * Find window and get a high level description of mouse location.
*/
                        wheremouse = FindWindow(myevent->where,
                        &whichwindow);
                        switch (wheremouse) {
                              case inDesk:
                              /*
                              * in top level window
                              */
                              ..... ; break;

                                    case inMenuBar:
                               /*
                              * in menu bar
                              */
                              ......; break;

                              case inGrow: case inDrag:
                              /*
                              *in window borders to manipulate the window
                              */
                              ...... ; break;

                                    case inContent:
                                    /*
                                    * in the content part of the window.
                                    * switch from global mouse coordinates to
                                    * coordinates relative to the window
                                    */
                                    localwhere = &myevent->where;
                                    GlobalToLocal (&localwhere);
```

Explicit Main Event Loop cont.

- Program cont.

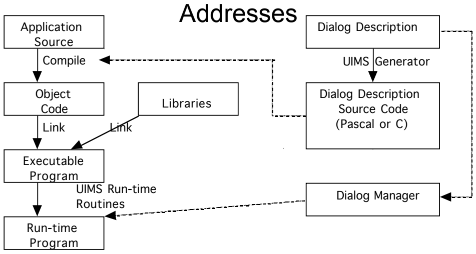
```
                              whereincontrol = FindControl (localwhere,
                                          &whichwindow, &whichcontrol);

                              /*
                              test whether the event happened in a control.
                              if so,findout where in the control.
                              */
                              if (whichcontrol != NIL) {
                              switch (whereincontrol) {
                              case inButton:
                              /*
                              * toolkit perform the lexical feedback
                              * until the end user releases the mouse
                                 * button.
                              */
                                 if (TrackControl(whichcontrol,
                                          localwhere, ...) )
                                 ....... ;
                                 break;
                                 ... ; } /* end switch (whereincontrol) */

                           /* end if (whichcontrol != NIL) */

                  } /* end switch (wheremouse) */

            } /* end switch (myevent->what) */

      } /* end while (go) */
```

Big Problem: Hooking the UIMS and application back together

- How does the UIMS send the application the information to process the correct semantics for an event?
  - Can associate application procedures directly by name
    - Kernel models
  - Can associate application procedures through callbacks
    - Client-server models, e.g. Motif

UIMS to Application Semantics

Associating Procedure Names or Addresses

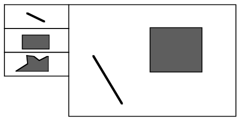


UIMS to Application Semantics

Example: Simple Drawing Application

Application Semantics

```
if line-icon,
   DrawLine(X1,Y1,X2,Y2)
if rect-icon,
   DrawRect(X1,Y1,X2,Y2)
if poly-icon,
  do:
  get X,Y points
  StartPoly(X,Y)
  AddPolyPoint(X,Y)
  if poly-complete,
        EndPoly( )
```



UIMS to Application Semantics

Associating Procedure Names

- In the application program, the command is associated with a procedure name and event record

```
Procedure DoSemanticCommand (CommandNum:Integer; Evnt:
   EventRecord);
     Begin
          Case CommandNum Of
               0: DeleteLine(Evnt);
               1: DrawLine(Evnt);
               2: DeleteCircle(Evnt);
               3: DrawCircle(Evnt);
               4: QuitProg(Evnt);
          End;
     End; {DoSemanticCommand}
```

UIMS to Application Semantics

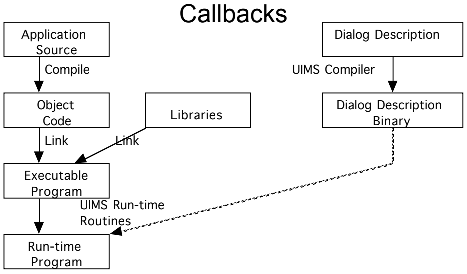
An Event Record

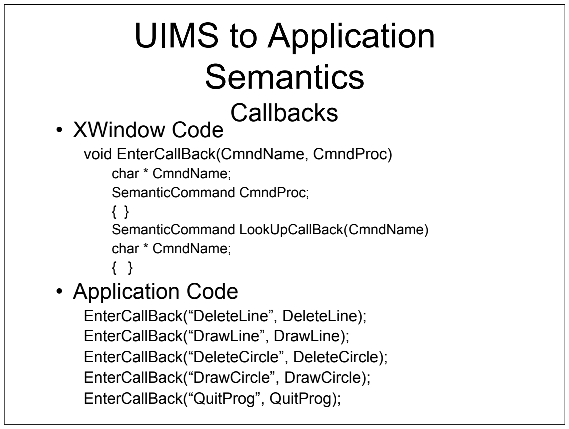
```
Event = Record
     EventCode: Integer;
     MouseX, MouseY:Integer;
     EventValue: Integer;
     Time:Integer;
End;
```

where EventCode “1” for mouse button;
EventValue “2” for down

UIMS to Application Semantics

Callbacks





UIMS to Application Semantics

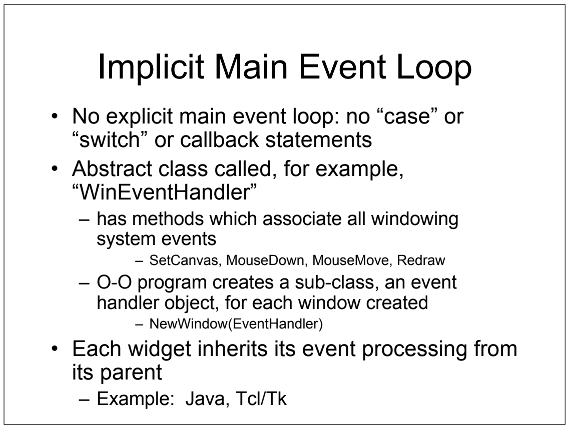
Callbacks

- XWindow Code

```
void EnterCallBack(CmndName, CmndProc)
    char * CmndName;
    SemanticCommand CmndProc;
    { }
    SemanticCommand LookUpCallBack(CmndName)
    char * CmndName;
    {  }
```

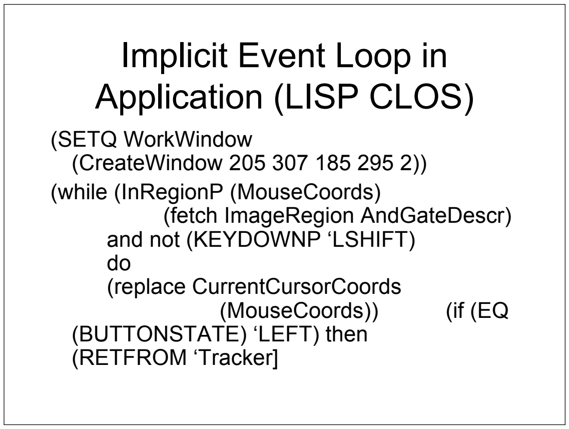
- Application Code

```
EnterCallBack(“DeleteLine”, DeleteLine);
EnterCallBack(“DrawLine”, DrawLine);
EnterCallBack(“DeleteCircle”, DeleteCircle);
EnterCallBack(“DrawCircle”, DrawCircle);
EnterCallBack(“QuitProg”, QuitProg);
```



Implicit Main Event Loop

- No explicit main event loop: no “case” or “switch” or callback statements
- Abstract class called, for example, “WinEventHandler”
  - has methods which associate all windowing system events
    - SetCanvas, MouseDown, MouseMove, Redraw
  - O-O program creates a sub-class, an event handler object, for each window created
    - NewWindow(EventHandler)
- Each widget inherits its event processing from its parent
  - Example: Java, Tcl/Tk



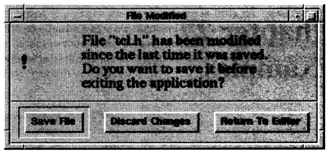
Implicit Event Loop in Application (LISP CLOS)

```
(SETQ WorkWindow
   (CreateWindow 205 307 185 295 2))
(while (InRegionP (MouseCoords)
              (fetch ImageRegion AndGateDescr)
        and not (KEYDOWNP ‘LSHIFT)
        do
        (replace CurrentCursorCoords
                    (MouseCoords))          (if (EQ
   (BUTTONSTATE) ‘LEFT) then
   (RETFROM ‘Tracker]
```

Implicit Main Event Loop Tcl/Tk

- Each Tk widget is a window
- Each widget has pre-defined event handlers
  - Example: Button widget responds to mouse button
- Can attach a Tcl script to an event handler to process application semantics for widget
  - Example: Bind command
- Other events in event queue
  - “after” generates timer event (used for animation, etc.)
  - “fileevent” when file descriptor becomes readable or writable
  - Process redraws after input events

Tcl/Tk Example



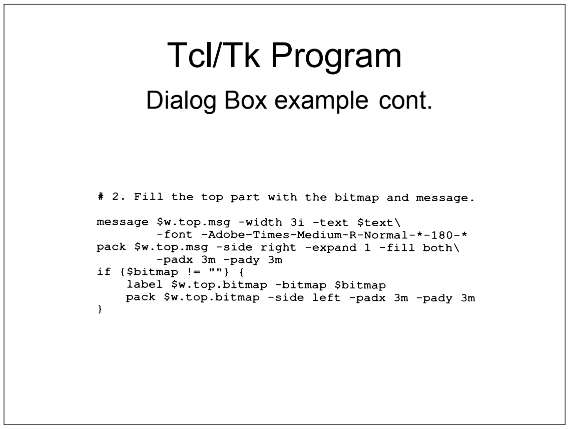
File "tcl.h" has been modified since the last time it was saved. Do you want to save it before exiting the application?

```
dialog .d {File Modified} {File "tcl.h" has been modified since\
    the last time it was saved. Do you want to save it before\
    exiting the application?} warning 0 {Save File} \
    {Discard Changes} {Return To Editor}
```

Tcl/Tk Program

Dialog Box example

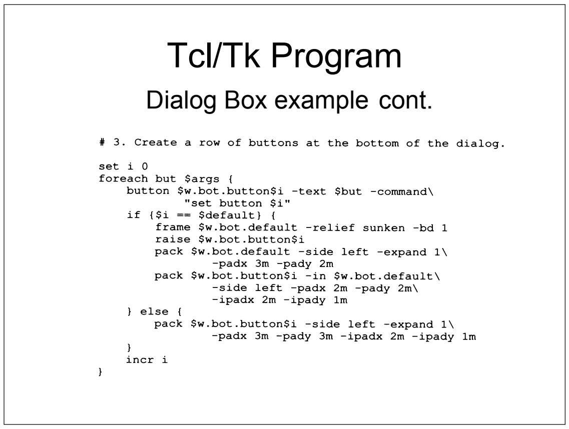
```
proc dialog {w title text bitmap default args} {
    global button

    # 1. Create the top-level window and divide it into top
    # and bottom parts.

    toplevel $w -class Dialog
    wm title $w $title
    wm iconname $w Dialog
    frame $w.top -relief raised -bd 1
    pack $w.top -side top -fill both
    frame $w.bot -relief raised -bd 1
    pack $w.bot -side bottom -fill both
```



Tcl/Tk Program

Dialog Box example cont.

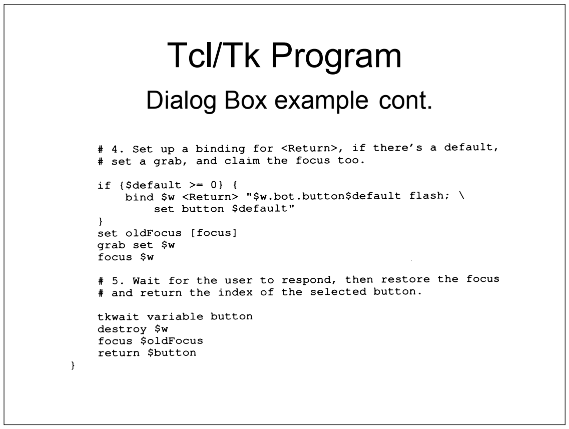
```
# 2. Fill the top part with the bitmap and message.

message $w.top.msg -width 3i -text $text\
        -font -Adobe-Times-Medium-R-Normal-*-180-*
pack $w.top.msg -side right -expand 1 -fill both\
        -padx 3m -pady 3m
if {$bitmap != ""} {
    label $w.top.bitmap -bitmap $bitmap
    pack $w.top.bitmap -side left -padx 3m -pady 3m
}
```



Tcl/Tk Program

Dialog Box example cont.

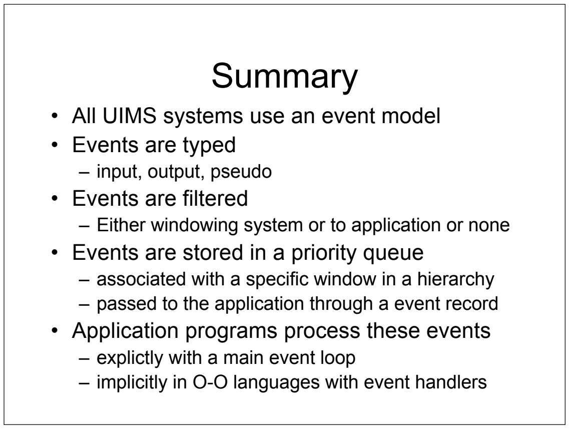
```
# 3. Create a row of buttons at the bottom of the dialog.

set i 0
foreach but $args {
    button $w.bot.button$i -text $but -command\
            "set button $i"
    if {$i == $default} {
        frame $w.bot.default -relief sunken -bd 1
        raise $w.bot.button$i
        pack $w.bot.default -side left -expand 1\
                -padx 3m -pady 2m
        pack $w.bot.button$i -in $w.bot.default\
                -side left -padx 2m -pady 2m\
                -ipadx 2m -ipady 1m
    } else {
        pack $w.bot.button$i -side left -expand 1\
                -padx 3m -pady 3m -ipadx 2m -ipady 1m
    }
    incr i
}
```



Tcl/Tk Program

Dialog Box example cont.

```
    # 4. Set up a binding for <Return>, if there’s a default,
    # set a grab, and claim the focus too.

    if {$default >= 0} {
        bind $w <Return> "$w.bot.button$default flash; \
            set button $default"
    }
    set oldFocus [focus]
    grab set $w
    focus $w

    # 5. Wait for the user to respond, then restore the focus
    # and return the index of the selected button.

    tkwait variable button
    destroy $w
    focus $oldFocus
    return $button
}
```



Summary

- All UIMS systems use an event model
- Events are typed
  - input, output, pseudo
- Events are filtered
  - Either windowing system or to application or none
- Events are stored in a priority queue
  - associated with a specific window in a hierarchy
  - passed to the application through a event record
- Application programs process these events
  - explictly with a main event loop
  - implicitly in O-O languages with event handlers