New Technical Notes

Macintosh

Developer Support

_ZoomWindow

Toolbox

Revised by: Craig Prouse April 1990
Written by: Jim Friedlander June 1986

This Technical Note contains some hints about using `_ZoomWindow`.

Changes since February 1990: Fixed a bug in `DoWZoom` which caused crashes if the content of a window did not intersect with any device's `gdRect`. Also made `DoWZoom` more robust by making `savePort` a local variable and checking for off-screen and inactive `GDevice` records. (One variable name has changed.) Additional minor changes: Corrected original sample code to use `_EraseRect` before zooming and added references to Human Interface Note #7, Who's Zooming Whom? for more subtle and application-specific considerations.

Basics

`_ZoomWindow` allows a window to be toggled between two states (where "state" means size and location): a default state and a user-selectable state. The default state stays the same unless the application changes it, while the user-selectable state is altered when the user changes the size or location of a zoomable window. The code to handle zoomable windows in a main event loop would look something like the examples which follow.

Note: `_ZoomWindow` assumes that the window that you are zooming is the current `GrafPort`. If `thePort` is not set to the window that is being zoomed, an address error is generated.

MPW Pascal

```
CASE myEvent.what OF
  mouseDown: BEGIN
    partCode:= FindWindow(myEvent.where, whichWindow);
      CASE partCode OF
        inZoomIn, InZoomOut:
          IF TrackBox(whichWindow, myEvent.where, partCode) THEN
            BEGIN
              GetPort(oldPort);
              SetPort(whichWindow);
              EraseRect(whichWindow^.portRect);
              ZoomWindow(whichWindow, partCode, TRUE);
              SetPort(oldPort);
            END; {IF}
```

```
    ...{and so on}
END; {CASE}
```

```
        END; {mouseDown}
      ...{and so on}
    END; {CASE}
```

MPW C

```
    switch (myEvent.what) {
      case mouseDown:
        partCode = FindWindow(myEvent.where, &whichWindow);
        switch (partCode) {
          case inZoomIn:
          case inZoomOut:
            if (TrackBox(whichWindow, myEvent.where, partCode)) {
              GetPort(&oldPort);
              SetPort(whichWindow);
              EraseRect(whichWindow->portRect);
              ZoomWindow(whichWindow, partCode, true);
              SetPort(oldPort);
              } /* if */
            break;
          ... /* and so on */
        } /* switch */
      ... /* and so on */
    } /* switch */
```

If a window is zoomable, that is, if it has a window definition `ID = 8` (using the standard `'WDEF'`), `WindowRecord.dataHandle` points to a structure that consists of two rectangles. The user-selectable state is stored in the first rectangle, and the default state is stored in the second rectangle. An application can modify either of these states, though modifying the user-selectable state might present a surprise to the user when the window is zoomed from the default state. An application should also be careful to not change either rectangle so that the title bar of the window is hidden by the menu bar.

Before modifying these rectangles, an application must make sure that `DataHandle` is not `NIL`. If it is `NIL` for a window with window definition `ID = 8`, that means that the program is not executing on a system or machine that supports zooming windows.

One need not be concerned about the use of a window with window definition `ID = 8` making an application machine-specific—if the system or machine that the application is running on doesn't support zooming windows, `_FindWindow` never returns `inZoomIn` or `inZoomOut`, so neither `_TrackBox` nor `_ZoomWindow` are called.

If `DataHandle` is not `NIL`, an application can set the coordinates of either rectangle. For example, the Finder sets the second rectangle (default state) so that a zoomed-out window does not cover the disk and trash icons.

For the More Adventurous (or Seeing Double)

Developers should long have been aware that they should make no assumptions about the screen size and use `screenBits.bounds` to avoid limiting utilization of large video displays. Modular Macintoshes and Color QuickDraw support multiple display devices, which invalidates the use of `screenBits.bounds` unless the boundary of only the primary display (the one with the menu bar) is desired. When dragging and growing windows in a **multi-screen environment**, developers are now urged to use the bounding rectangle of the

GrayRgn. In most cases, this is not a major modification and does not add a significant amount of code. Simply define a variable

```
desktopExtent := GetGrayRgn^^.rgnBBox;
```

and use this in place of screenBits.bounds. When zooming a document window, however, additional work is required to implement a window-zooming strategy which fully conforms with Apple's Human Interface Guidelines.

One difficulty is that when a new window is created with _NewWindow or _GetNewWindow, its default stdState rectangle (the rectangle determining the size and position of the zoomed window) is set by the Window Manager to be the gray region of the main display device inset by three pixels on each side. If a window has been moved to reflect a position on a secondary display, that window still zooms onto the main device, requiring the user to pan across the desktop to follow the window. The preferred behavior is to zoom the window onto the device containing the **largest portion** of the unzoomed window. This is a perfect example of a case where it is necessary for the application to modify the default state rectangle before zooming.

DoWZoom is a Pascal procedure which implements this functionality. It is a good example of how to manipulate both a WStateData record and the Color QuickDraw device list. On machines without Color QuickDraw (e.g., Macintosh Plus, Macintosh SE, Macintosh Portable) the stdState rectangle is left unmodified and the procedure reduces to five instructions, just like it is illustrated under "Basics." If Color QuickDraw is present, a sequence of calculations determines which display device contains most of the window prior to zooming. That device is considered dominant and is the device onto which the window is zoomed. A new stdState rectangle is computed based on the gdRect of the dominant GDevice. Allowances are made for the window's title bar, the menu bar if necessary, and for the standard three-pixel margin. (Please note that DoWZoom only mimics the behavior of the default _ZoomWindow trap as if it were implemented to support multiple displays. It does not account for the "natural size" of a window for a particular purpose. See Human Interface Note #7, Who's Zooming Whom?, for details on what constitutes the natural size of a window.) It is not necessary to set stdState prior to calling _ZoomWindow when zooming back to userState, so the extra code is not executed in this case.

DoWZoom is too complex to execute within the main event loop as shown in "Basics," but if an application is already using a similar scheme, it can simply add the DoWZoom procedure and replace the conditional block of code following

```
IF TrackBox…
```

with

```
DoWZoom(whichWindow, partCode);.
```

Happy Zooming.

```
PROCEDURE DoWZoom (theWindow: WindowPtr; zoomDir: INTEGER);
VAR
 windRect, theSect, zoomRect : Rect;
 nthDevice, dominantGDevice : GDHandle;
 sectArea, greatestArea : LONGINT;
 bias : INTEGER;
```

```
 sectFlag : BOOLEAN;
 savePort : GrafPtr;
BEGIN
 { theEvent is a global EventRecord from the main event loop }
 IF TrackBox(theWindow,theEvent.where,zoomDir) THEN
   BEGIN
     GetPort(savePort);
     SetPort(theWindow);
     EraseRect(theWindow^.portRect);    {recommended for cosmetic reasons}

     { If there is the possibility of multiple gDevices, then we  }
     { must check them to make sure we are zooming onto the right }
     { display device when zooming out. }
     { sysConfig is a global SysEnvRec set up during initialization  }
     IF (zoomDir = inZoomOut) AND sysConfig.hasColorQD THEN
       BEGIN
         { window's portRect must be converted to global coordinates }
         windRect := theWindow^.portRect;
         LocalToGlobal(windRect.topLeft);
         LocalToGlobal(windRect.botRight);
         { must calculate height of window's title bar }
         bias :=   windRect.top - 1
             -  WindowPeek(theWindow)^.strucRgn^^.rgnBBox.top;
         windRect.top := windRect.top - bias; {Thanks, Wayne!}
         nthDevice := GetDeviceList;
         greatestArea := 0;
         { This loop checks the window against all the gdRects in the   }
         { gDevice list and remembers which gdRect contains the largest }
         { portion of the window being zoomed. }
         WHILE nthDevice <> NIL DO
           IF TestDeviceAttribute(nthDevice,screenDevice) THEN
             IF TestDeviceAttribute(nthDevice,screenActive) THEN
               BEGIN
                 sectFlag := SectRect(windRect,nthDevice^^.gdRect,theSect);
                 WITH theSect DO
                   sectArea := LONGINT(right - left) * (bottom - top);
                 IF sectArea > greatestArea THEN
                   BEGIN
                     greatestArea := sectArea;
                     dominantGDevice := nthDevice;
                   END;
                 nthDevice := GetNextDevice(nthDevice);
               END; {of WHILE}
         { We must create a zoom rectangle manually in this case. }
         { account for menu bar height as well, if on main device }
         IF dominantGDevice = GetMainDevice THEN
           bias := bias + GetMBarHeight;
         WITH dominantGDevice^^.gdRect DO
           SetRect(zoomRect,left+3,top+bias+3,right-3,bottom-3);
         { Set up the WStateData record for this window. }
         WStateDataHandle(WindowPeek(theWindow)^.dataHandle)^^.stdState := zoomRect;
       END; {of Color QuickDraw conditional stuff}

     ZoomWindow(theWindow,zoomDir,TRUE);
     SetPort(savePort);
   END;
END;
```

In an attempt to avoid declaring additional variables, the original version of this document was flawed. In addition, the assignment statement responsible for setting the `stdState` rectangle

is relatively complex and involves two type-casts. The following may look like C, but it really is Pascal. Trust me.

```
WStateDataHandle(WindowPeek(theWindow)^.dataHandle)^^.stdState := zoomRect;
```

It could be expanded into a more readable form such as:

```
VAR
   theWRec : WindowPeek;
   zbRec   : WStateDataHandle;

theWRec := WindowPeek(theWindow);
zbRec := WStateDataHandle(theWRec^.dataHandle);
zbRec^^.stdState := zoomRect;
```

Further Reference:

- *Inside Macintosh*, Volume IV, The Window Manager (pp. 49–52)
- *Inside Macintosh*, Volume V, Graphics Devices (p. 124), The Window Manager (p. 210)
- Human Interface Note #7, Who's Zooming Whom?