New Technical Notes

Macintosh

®

DV 19 - Drivers & DAs in Need of (a Good) Time
Devices

Revised by: Pete Helme   October 1989
Written by: Pete Helme   August 1989

This Technical Note describes a few complications which rear their rather ugly little heads when a desk accessory or driver needs periodic time. It also presents a few solutions to work around these problems and make life easier, at least periodically.
Changes since August 1989: Corrected `_BitClr` and `_BitSet` examples. Okay, I admit it. I was having too good of a time when I wrote the original Note and messed up the bit manipulations at the end. My vision was blurred; I was in no condition to see those tiny little things.

See Jane's Heap, See accRun…

MultiFinder is our friend. Our friend, that is, until a driver or desk accessory is called when in an unknown heap. Then things get complicated. When a driver is called at `accRun` time under MultiFinder, one can never be exactly sure of the heap in which it will find itself. When a DA receives an open call, or any other messages besides accRun, under MultiFinder, the system heap is switched in as the current heap.* (*NOTE: This is true unless a user "force" switched the DA into an application heap by holding down the Option key when opening the DA. In this particular case, the application's heap will be swtiched in.)

During the `accRun` cycle, whatever heap is currently switched in will be the driver's heap as well, and surprise, surprise, that heap may not be the system heap.

This situation could be a real problem if your DA allocates memory or creates a window during that `accRun` period. Why? What if the application whose heap the DA is in suddenly slips a bit and decides to call it quits before the DA? You'd be stuck with allocated blocks in a zone that suddenly doesn't exist. Eventually, your DA would go belly up, and whoever bought your DA or driver would be on the phone to a local dealer demanding retribution.

So what's the solution? The easiest way out of this situation is to simply not do any memory allocation or display any newly created windows or dialog boxes during `accRun`. So what if it's a cop out, it's easy to implement.

Being the good souls that we are in DTS, we're not going to leave you hanging there with nowhere to go. We prefer you heed the previous solution, but we realize that there may be rare times when you might need a window during `accRun`. We've devised a solution, albeit a bit strange, but one that's easy enough to use.

The basic problem is that the DA needs to know in which heap it should be allocating it's new storage. It would be nice if the DA knew in which heap it was opened and could allocate the new stuff there, and it's easy enough to do, so here is what you need to know to do it.

Switching from the current heap to the "preferred" heap is fairly simple. When you feel the need to allocate memory or create a window during `accRun`, first save the current heap zone with `_GetZone`. Now, get the handle to the actual driver for the DA. You can do this by looking at the `dCtlDriver` offset of the DAs device control entry (DCE). The DCE is always in register `A1` when a control call to the DA is made. Use `_HandleZone` on the handle to the DAs driver to give you a pointer to the heap in which the driver resides. Pass that value to `_SetZone`. Once you have switched in the correct heap, do whatever memory allocation or window creation you need, and then make sure to set the current zone back to the saved zone with `_SetZone`.

The following short routine, borrowed, in part, from an MPW sample DA, shows one way to set up the correct zone.

```
pascal short DRVRControl(CntrlParam *ctlPB, DCtlPtr dCtl)
{
        extern  void    doCtlEvent();
        extern  void    doPeriodic();

        THz     driverZone;
        THz     savedZone;

        /*
         * The current grafPort is saved & restored by the Desk Manager
         */
        switch (ctlPB->csCode) {
                case ACCEVENT:                          /* accEvent */
                    HLock(dCtl->dCtlStorage);   /* Lock handle since it will
                                                        be dereferenced */
                    doCtlEvent( *((EventRecord **) &ctlPB->csParam[0]),
                                (Globals *)(*dCtl->dCtlStorage));
                    HUnlock(dCtl->dCtlStorage);
                    break;

                /*
                 * Hey!  Look here!
                 */
                case ACCRUN:                            /* periodicEvent */
                        savedZone = GetZone();          /* save a pointer to current heap
                                                        */
                        driverZone = HandleZone(dCtl->dCtlDriver); /* get the heap our
                                                                        driver resides
                                                                        in */
                        SetZone(driverZone);            /* use that as the current heap */
                        doPeriodic(dCtl);               /* go do your periodic stuff */
                        SetZone(savedZone);             /* restore the old heap */
                        break;

                default:
                        break;
        }
        return 0;
}
```

DV 19 - Drivers & DAs in Need of (a Good) Time

Devices

One note of caution: Watch out for changes in the resource chain when in `accRun`, as it may not be what you expect when MultiFinder is active.

"Houston, We've Got a Re-Entry Problem"

Displaying an alert or other modal dialog box is a common occurrence in Macintosh programming, even in DAs. But since DAs are not applications, modal dialog boxes pose other problems when displayed under MultiFinder. This problem is reentrancy. If your DA or driver asks for periodic time, it continues to receive it when it display a modal dialog box. Bummer. Your modal dialog routine might even be called again, and again, and again, and again, and you get the idea. This problem occurs because `_ModalDialog` calls the `_SystemTask` trap, which in turn calls drivers which asked for time, including yours. There is no internal check by the System for this possible problem, so it's up to you and your driver to be prepared.

We realize that some DAs and drivers expect, and depend upon, this functionality. We're just taking this opportunity to inform the rest of you that this is a situation about which you should be aware.

An easy way to avoid this issue is to simply tell the Device Manager not to call your DA when you display an alert or other modal dialog box. Remember that `dNeedTime` bit you set when you opened your DA so you'd get time? Just clear it before your call to `_Alert` or `_ModalDialog`. As long as the bit is clear, your DA does not receive any periodic time. Remember to reset it once you are done with your `_Alert` or `_ModalDialog` trap call.

The `_BitClr` and `_BitSet` Toolbox utilities are a mite on the brain-damaged side, and the bits are the reverse of conventional 680x0 numbering (numbering starts from the high-order bit instead of the low-order bit). This difference necessitates a calculation for figuring out the correct bit as shown in the following example: (I think whoever wrote these Toolbox utilities did this just to see if anyone was paying attention.)

Pascal

```
BitClr(@dce^.dCtlFlags, 2);          { clear bit 5/dNeedTime bit. IM I-471 }
BitSet(@dce^.dCtlFlags, 2);          { set bit 5/dNeedTime bit.  IM I-471 }
```

or the kind of more efficient, but less efficient than C:

```
CONST
dNeedTime = $2000;                   { Bit 5 of high-order byte of word }

dce^.dCtlFlags := BAND(dce^.dCtlFlags, BNOT(dNeedTime)); { clear bit 5/dNeedTime
                                                           bit. }
dce^.dCtlFlags := BOR(dce^.dCtlFlags, dNeedTime);        { set bit 5/dNeedTime bit.
}
```

C

```
BitClr(&dce->dCtlFlags, 2);          /* clear bit 5/dNeedTime bit. IM I-471 */
BitSet(&dce->dCtlFlags, 2);          /* set bit 5/dNeedTime bit.  IM I-471 */
```

or the somewhat more efficient:

Devices

```
#define dNeedTime 0x2000          /* Bit 5 of high-order byte of word */

dce->dCtlFlags &= ~dNeedTime;     /* clear bit 5/dNeedTime bit. */
dce->dCtlFlags |= dNeedTime;      /* set bit 5/dNeedTime bit.  */
```

One More Thing…

We cannot overemphasize our viewpoint that if you are writing a DA and the result looks and acts more like an application, then **write an application** instead and save us all a lot of headaches.

Further Reference:

- *Inside Macintosh*, Volume II, The Memory Manager
- Technical Note TB 35 - MultiFinder Miscellanea