New Technical Notes

Macintosh

Developer Support

Device Driver Q&As

Devices

Revised by: Developer Support Center June 1993
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&A in this Technical Note:
Backlight driver save brightness control call

Backlight driver save brightness control call
Date Written: 1/14/93
Last reviewed: 4/1/93

When I make a control call to the Backlight driver to save brightness (0x4302) for a Macintosh PowerBook Duo, the return is controlErr (-17). How can I change the brightness and have it stick after calling the cdev Active to release backlight control?

—

The "save brightness" call doesn't exist for models with user *hardware* control. The main reason for the call is to save the backlight setting in PRAM for models with software-only control. On fly-by-wire backlight controls (defined as hardware control with a slider or button controls such as on the PowerBook 140, 170, and Duo), this function doesn't make sense since the hardware will dictate the boot value. As a result, that control call will return an error.

Calling one Macintosh driver from another

Date Written: 6/3/92
Last reviewed: 9/15/92

Is it OK for my SCSI driver's prime routine to call another SCSI driver's prime routine? The original call to my driver would most likely be caused by an asynchronous or synchronous _Read or _Write. My driver's prime routine might call another driver (never my own) with a _Read sync or _Write sync (never async) before it completes. My driver always completes synchronously. Since the Device Manager maintains a seperate queue for each driver shouldn't this be OK?

—

This should work fine from the Device Manager's standpoint; there shouldn't be any reason why the calls should run into any problems on that level. However, depending on which drivers you're calling, there might be problems with whether the current machine will allow the request to complete. For example, let's say your driver is a serial replacement driver, and it's making calls to a SCSI disk driver to a huge buffer. If someone were to call a traditional serial driver at interrupt time asynchronously, it could be expected to work regardless of the state of the machine; the request is queued and handled as characters come in later. If, however, someone were to make a call to your driver at a strange time (say from a Time Manager task), your driver may not be able to satisfy the request because it needs to talk to a SCSI disk driver and the SCSI bus may be locked up or for any number of other reasons, the SCSI driver can't respond.

Situations like this are all you need to look out for; the Device Manager shouldn't give you any trouble. If the driver being called doesn't depend on system resources that might be unavailable for any reason, including interrupt level, then you should be OK. For example, an IAC driver which merely communicated blocks of data back and forth should easily be able to satisfy all requests; this situation wouldn't be a problem then.

Installing a Macintosh driver at INIT time
Date Written: 5/18/92
Last reviewed: 7/13/92

Examining the sample code on several Developer CDs, it seems that there are two approaches in installing the driver in INIT time:

1. Search for an empty slot in the unit entry table, fail if it's full;
2. Search for an empty slot in the unit entry table, try to resize it if it's full;

What's the official approach to installing a driver at INIT time? Is there an official code sample showing how to install a driver at INIT time?

—

The correct approach is to resize the unit table if it is full; if a piece of sample code does not resize the table when it needs it, it's just being lazy.

How Macintosh Standard File determines device type

Date Written: 3/6/92
Last reviewed: 8/1/92

Is there a way besides using csCode 23 to determine the type of the driver from the volume's parameter block?

___

There's no way of knowing for sure whether a device is a floppy, SCSI, AppleShare or whatever. Standard File gets the DCTL entry for the driver for a given device and examines the driver name. If the driver name is .Sony, for instance, you are dealing with a floppy disk. The routine below returns the proper device type based on the Standard File's method of doing it. The routine includes CD-ROMs, and defaults to hard disk if it can not identify the driver as something else.

Disk driver names can be:

```
.Sony         3.5 floppy drive driver
.SCSI0x       some type of SCSI disk, probably hard disk
.AppleCD      Apples CD ROM driver
.AFPTrans…    AFP translator (Appleshare)
```

These are the standard Apple names. Third-party disk devices may not conform to this standard.

```
Function GetVolType(paramBlock:HParmBlkPtr):Integer;

var    theDCTL:DCtlHandle;
   thePtr:stringPtr;
   theType:Integer;
   barney:Longint;

begin
   theDctl:=GetDCTLEntry(paramBlock^.ioVDRefNum);
   if BTst(theDctl^^.dCtlFlags,6) then
     thePtr:=StringPtr(handle(theDctl^^.dctlDriver)^)
   else
     thePtr:=StringPtr(theDctl^^.dctlDriver);
   thePtr:=StringPtr(ord4(thePtr)+18);
   theType:=GenericHD;
   if Copy(thePtr^,2,4) = 'Sony' then
     theType:=GenericFloppy;
   if Copy(thePtr^,2,4) = 'AFPT' then
     theType:=GenericServer;
   if thePtr^='.AppleCD' then
     theType:=GenericCDRom;
   GetVolType:=theType;
end;
```

Looking up Macintosh drivers installed from other applications

Date Written: 3/24/92
Last reviewed: 8/1/92

Is there a way for a Macintosh application to install its driver into the system unit table so it will be available from all applications and after the driver exits?

—

The Process Manager doesn't remove the driver's entry when processes are switched if the driver itself is located in the system heap. It then assumes this is a "system-wide" driver, and the driver appears in all applications.

Macintosh interfaces for Apple Scanner
Date Written: 1/15/91
Last reviewed: 3/25/91

Is there a set of Macintosh C interfaces for the Apple Scanner driver?

—

The Apple Scanner Reference (APDA #M7078) comes with a disk that has interface files for assembly language, Pascal, and C. The C headers are in a file called ScIntf.j. Just #include this file and Link with the file called ScIntf.a.o (on that same disk), and you should be able to control the scanner from C.

How to make sure inactive application VBL tasks always get time
Date Written: 12/21/90
Last reviewed: 1/16/91

Why is my Macintosh VBL task (installed via SlotVBInstall in VBL queue) getting dumped after a few successful operations if I use a GetNextEvent or EventAvail call within the routine, but runs OK if I don't?

—

One of the lesser-known mysteries of running in the world of MultiFinder is that on either major or minor context switches (which both GetNextEvent and EventAvail allow), VBL tasks installed in the application heap will be unhooked until that application is the current application again.

If you wish for your VBL task to always get time install the VBLRecord in the system heap. One way to do this would be to use NewPtrSys(sizeof(VBLRecord)) and then to cast the resulting pointer as needed.

System 6 "Faceless Background Driver" bug and workaround
Date Written: 12/3/90
Last reviewed: 12/19/90

We want to improve performance by getting frequent and consistent run times, but our Macintosh driver seems to get run time only about every 0.5 sec under MultiFinder, even when set to be awakened every tic. This current behavior degrades our performance noticeably when Finder is the active application and no other applications are running. Is there a way around this problem?

—

We successfully reproduced the result you reported. With only the Finder running, our driver got called on once every $1E (30) ticks. It seems that the reason for this is considered a

MultiFinder bug (called the “Faceless Background Driver” bug). What happens is that MultiFinder doesn’t call SystemTask when all applications are asleep. When Finder is the the only application, for example, it gets awakened once every 30 ticks. Otherwise, it’s asleep. When it’s asleep, MultiFinder just spins its gears, and fails to call SystemTask. If you run another application that has a much smaller sleep time (like AppleLink), you’ll find that your driver gets called more frequently.

Someone in a situation almost identical to yours wrote a VBL that would make a special Control call to the driver. The code that responded to the Control call would do whatever processing he wanted to do, taking care not to move or allocate any memory. Then, the next time he got an accRun message, he'd take care of any memory munging that he couldn't do at VBL time. He says that it wasn't hard to do, and that it all worked out very well.

Otherwise, we don't know if there is much you can do. System 7.0 is supposed to fix the problem, but we haven't double-checked that yet. It also doesn't help all your System 6.0.x customers, so you might want to investigate the VBL solution.