Apple II Technical Notes

Developer Technical Support

ProDOS 8
#21: Identifying ProDOS Devices

Revised by: Dave Lyons & Matt Deatherage March 1990
Written by: Matt Deatherage & Dan Strnad November 1988

This Technical Note describes how to identify ProDOS devices and their characteristics given the ProDOS unit number. This scheme should only be used under ProDOS 8.

Changes since January 1990: Modified AppleTalk call code for compatibility with ProDOS 8 versions earlier than 1.5 and network-booted version 1.4.

There are various reasons why an application would want to identify ProDOS devices. Although ProDOS itself takes great pains to treat all devices equally, it has internal drivers for two types of devices: Disk II drives and the /RAM drive provided on 128K or greater machines. Because all devices really are not equal (i.e., some cannot format while others are read-only, etc.), a developer may need to know how to identify a ProDOS device.

Although the question of how much identification is subjective for each developer, ProDOS 8 offers a fair level of identification; the only devices which cannot be conclusively identified are those devices with RAM-based drivers, and they could be anything. The vast majority of ProDOS devices can be identified, however, so you could prompt the user to insert

a disk in UniDisk 3.5 #2, instead of Slot 2, Drive 2, which could be confusing if the user has a IIc or IIGS.

Note that for the majority of applications, this level of identification is unnecessary. Most applications simply prompt the user to insert a disk by its name, and the user can place it in any drive which is capable of working with the media of the disk. You should avoid requiring a certain disk to be in a specific drive since doing so defeats much of the device-independence which gives ProDOS 8 its strength.

When you do need to identify a device (i.e., if you need to format media in a Disk II or /RAM device), however, the process is fairly straightforward. This process consists of a series of tests, any one of which could end with a conclusive device identification. It is not possible to look at a single ID byte to determine a particular device type. You may determine rather quickly that a device is a SmartPort device, or you may go all the way through the procedure to identify a third-party network device. For those developers who absolutely must identify devices, DTS presents the following discussion.

Isn't There Some Kind of "ID Nibble?"

ProDOS 8 does not support an "ID nibble." Section 5.2.4 of the *ProDOS 8 Technical Reference Manual* states that the low nibble of each unit number in the device list "is a device identification: 0 = Disk II, 4 = Profile, $F = /RAM."

When ProDOS 8 finds a “smart” ProDOS block device while doing its search of the slots and ports, it copies the high nibble of $CnFE (where n is the slot number) into the low nibble of the unit number in the global page. The low nibble then has the following definition:

Bit 3: Medium is removable
Bit 2: Device is interruptible
Bit 1-0: Number of volumes on the device (minus one)

As you can see, it is quite easy for the second definition to produce one of the original values (e.g., 0, 4, or $F) in the same nibble for completely different reasons. You should ignore the low nibble in the unit number in the global page when identifying devices since the first definition is insufficient to uniquely identify devices and the second definition contains no information to specifically identify devices. Once you do identify a ProDOS block device, however, you may look at $CnFE to obtain the information in the second definition above, as well as information on reading, writing, formatting, and status availability.

When identifying ProDOS devices, start with a list of unit numbers for all currently installed disk devices. As you progress through the identification process, you identify some devices immediately, while others must wait until the end of the process for identification.

Starting with the Unit Number

ProDOS unit numbers (`unit_number`) are bytes where the bits are arranged in the pattern DSSS0000, where D = 0 for drive one and D = 1 for drive two, SSS is a three-bit integer with values from one through seven indicating the device slot number (zero is not a valid slot number), and the low nibble is ignored.

To obtain a list of the unit numbers for all currently installed ProDOS disk devices, you can perform a ProDOS MLI `ON_LINE` call with a unit number of $00. This call returns a unit number and a volume name for every device in the device list. ProDOS stores the length of the volume name in the low nibble of the unit number which `ON_LINE` returns; if an error occurs, the low nibble contains $0 and the byte immediately following the unit number contains an error code. For more information on the `ON_LINE` call, see section 4.4.6 of the *ProDOS 8 Technical Reference Manual*. A more detailed discussion of the error codes follows later in this Note.

To identify the devices in the device list, you need to know in which physical slot the hardware resides, so you can look at the slot I/O ROM space and check the device’s identification bytes. Note that the slot-number portion of the unit number does not always represent the physical slot of the device, rather, it sometimes represents the logical slot where you can find the address of the device’s driver entry point in the ProDOS global page. For example, if a SmartPort device

interface in slot 5 has more than two connected devices, the third and fourth devices are mapped to slot 2; this mapping gives these two devices unit numbers of $20 and $A0 respectively, but the device's driver entry point is still in the $C5xx address space.

ProDOS 8 Technical Note #20, Mirrored Devices and SmartPort, discusses this kind of mapping in detail. It also presents a code example which gives you the correct device-driver entry point (from the global page) given the unit number as input. Here is the code example from that Note for your benefit. It assumes the `unit_number` is in the accumulator.

```
devcnt      equ   $BF31
devlst      equ   $BF32
devadr      equ   $BF10
devget      sta   unitno            ; store for later compare instruction
            ldx   devcnt            ; get count-1 from $BF31
devloop     lda   devlst,x          ; get entry in list
            and   #$F0              ; mask off low nibble
devcomp     cmp   unitno            ; compare to the unit_number we filled in
            beq   goodnum           ;
            dex
            bpl   devloop           ; loop again if still less than $80
            bmi   badunitno         ; error: bad unit number
goodnum     lda   unitno            ; get good copy of unit_number
            lsr   a                 ; divide it by 8
            lsr   a                 ; (not sixteen because devadr entries are
            lsr   a                 ; two bytes wide)
            tax
            lda   devadr,x          ; low byte of device driver address
            sta   addr
            lda   devadr+1,x        ; high byte of device driver address
            sta   addr+1
            rts
addr        dw    0                 ; address will be filled in here by goodnum
unitno      dfb   0                 ; unit number storage
```

Warning: Attempting to construct the device-driver entry point from the unit number is very dangerous. **Always** use the technique presented above.

Network Volumes

AppleTalk volumes present a special problem to some developers since they appear as "phantom devices," or devices which do not always have a device driver installed in the ProDOS global page. Fortunately, the ProDOS Filing Interface (PFI) to AppleTalk provides a way to identify network volumes through an MLI call. The ProDOS Filing Interface call `FIListSessions` is used to retrieve a list of the current sessions being maintained through PFI and any volumes mounted for those sessions.

In the following example, note the check for ProDOS 8 version 1.5 or higher, and the simulation of a `bad command` error under older versions (the $42 call under ProDOS 8 version 1.4 always crashes if ProDOS was launched from a local disk):

```
Network     LDA   #$04              ;require at least ProDOS 8 1.4
            CMP   $BFFF             ;KVERSION (ProDOS 8 version)
            BEQ   MoreNetwork       ;have to check further
            LDA   #$01              ;simulate bad command error
            BCS   ERROR             ;if 3 or less, no possibility of network
            BCC   NetCall           ;otherwise, try the network call

MoreNetwork LDA   $BF02             ;high byte of the MLI entry point
            AND   #$F0              ;strip off the low nibble
            CMP   #$C0              ;is the entry into the $Cn00 space?
            BEQ    NetCall          ;yes, so try AppleTalk
            LDA   #$01
            SEC
            BCS   ERROR             ;simulate bad command error

NetCall     JSR   $BF00             ;ProDOS MLI
            DFB   $42               ;AppleTalk command number
```

```
        DW      ParamAddr           ;Address of Parameter Table
        BCS     ERROR               ;error occurred
```

```
ParamAddr     DFB    $00                ;Async Flag (0 means synchronous only)
                                        ;note there is no parameter count
              DFB    $2F                ;command for FIListSessions
              DW     $0000              ;AppleTalk Result Code returned here
              DW     BufLength          ;length of the buffer supplied
              DW     BufPointer         ;low word of pointer to buffer
              DW     $0000              ;high word of pointer to buffer
                                        ;(THIS WILL NOT BE ZERO IF THE BUFFER IS
                                        ;NOT IN BANK ZERO!)
              DFB    $00                ;Number of entries returned here
```

If the `FIListSessions` call fails with a bad command error ($01), then AppleShare is not installed; therefore, there are no networks volumes mounted. If there is a network error, the accumulator contains $88 (Network Error), and the result code in the parameter block contains the specific error code. The list of current sessions is placed into the buffer (at the address `BufPointer` in the example above), but if the buffer is not large enough to hold the list, it retains the maximum number of current sessions possible and returns an error with a result code of $0A0B (`Buffer Too Small`). The buffer format is as follows:

```
SesnRef       DFB    $00                ;Sessions Reference number (result)
UnitNum       DFB    $00                ;Unit Number (result)
VolName       DS     28                 ;28 byte space for Volume Name
                                        ;(starts with a length byte)
VolumeID      DW     $0000              ;Volume ID (result)
```

This list is repeated for every volume mounted for each session (the number is placed into the last byte of the parameter list you passed to the ProDOS MLI). For example, if there are two volumes mounted for session one, then session one is listed two times. The `UnitNum` field contains the slot and drive number in unit-number format, and note that bit zero of this byte is set if the volume is a user volume (i.e., it contains a special "users" folder). This distinction is unimportant for identifying a ProDOS device as a network pseudo-device, but it is necessary for applications which need to know the location of the user volume. Note that if you mount two servers or more with each having its own user volume, the user volume found first in the list (scanned top to bottom) returned by `FIListSessions` specifies the user volume that an application should use. See the *AppleShare Programmer's Guide for the Apple IIGS* for more information on programming for network volumes.

If you keep a list of all unit numbers returned by the `ON_LINE` call and mark each one "identified" as you identify it, keep in mind that the unit numbers returned by `FIListSessions` and `ON_LINE` have different low nibbles which should be masked off before you make any comparisons.

Note: You should mark the network volumes as identified and **not** try to identify them further with the following methods.

What Slot is it Really In?

Once you have the address of the device driver's entry point and know that the device is not a network pseudo-device, you can determine in what physical slot the device resides. If the high byte of the device driver's entry point is of the form $Cn, then n is the physical slot number of the device. A SmartPort device mirrored to slot 2 has a device driver address of $C5xx, giving 5 as the physical slot number.

If the high byte of the device driver entry point is **not** of the form $Cn, then there are three other possibilities:

- The device is a Disk II with driver code inside ProDOS.
- The device is either /RAM with driver code inside ProDOS or a third-party auxiliary-slot RAM disk device with driver code installed somewhere in memory.
- The device is not a RAM disk but has a RAM-based device driver, like a third-party network device.

Auxiliary-slot RAM disks are identified by convention. Any device in slot 3, drive 2 (unit number $B0) is assumed to be an auxiliary-slot RAM disk since ProDOS 8 does not recognize any card which is not an 80-column card in slot 3 (see ProDOS 8 Technical Note #15, How ProDOS 8 Treats Slot 3). There is a chance that some other kind of device could be installed with unit number $B0, but it is not likely.

To identify various kinds of auxiliary-slot RAM disks, you must obtain the unit number from the ProDOS global page. The list of unit numbers starts at $BF32 (`DEVLST`) and is preceded by the number of unit numbers minus one (`DEVCNT`, at $BF31). You should search through this list until you find a unit number in the form $Bx; if the unit number is $B3, $B7, $BB, or $BF, you can assume the device to be an auxiliary-slot RAM disk which uses the auxiliary 64K bank of memory present in a 128K Apple IIe or IIc, or a IIGS. If the unit number is one of the four listed above, you must remove this device to safely access memory in the auxiliary 64K bank, but if the unit number is **not** one of the four listed above, you can assume the device to be an auxiliary-slot RAM disk which does **not** use the normal bank of auxiliary memory. (Some third-party auxiliary-slot cards contain more than one 64K auxiliary bank; the normal use of this memory is as a RAM disk. If the RAM-based driver for this kind of card does not use the normal auxiliary 64K bank for storage, it should have a unit number other than one of the four listed above.) If the unit number is not one of the four listed above, you may safely access the auxiliary bank of memory without first removing this device.

Section 5.2.2.3 of the *ProDOS 8 Technical Reference Manual* contains a routine which disconnects the appropriate RAM disk devices in slot 3, drive 2, without removing those drivers which do not use that bank, to allow use of the auxiliary 64K bank.

Note: Previous information from Apple indicated that /RAM could be distinguished from third-party RAM disks by a driver address of $FF00. Although the address has not changed, some third-party drivers may have addresses of $FF00 as well, although this is **not** supported. /RAM always has a driver address of $FF00 and unit number $BF, although any third-party RAM disk could install itself with similar attributes.

For Disk II devices, the three-bit slot number portion of the `unit_number` is **always** the

physical slot number. Disk II devices can never be mirrored to another slot (the Disk II driver does not support it); therefore, it is in the physical slot represented in the unit number which ProDOS assigns when it boots.

If the high byte of the device driver's entry point is not of the form $Cn, then you should assume that the slot number is the value SSS in the unit number (this is equivalent to assuming the device is a Disk II) for the next step, which is checking the I/O space for identification bytes.

What to Do With the Slot Number

Once you have the slot number, you can look at the slot I/O ROM space to determine the kind of device it is. As described in the *ProDOS 8 Technical Reference Manual*, ProDOS looks for the following ID bytes in ROM to determine if a ProDOS device is in a slot:

$Cn01 = $20
$Cn03 = $00
$Cn05 = $03

If you use the slot number, n, you obtained above, and the three values listed above are **not** present, then the device has a RAM-based driver and cannot further be identified.

If the three values previously discussed are present, then examination of $CnFF gives more information. If $CnFF = $00, the device is a Disk II. If $CnFF is any value other than $00 or $FF ($FF signifies a 13-sector Disk II, which ProDOS does not support), the device is a ProDOS block device.

For ProDOS block devices, the byte at $CnFE contains several flags which further identify the device; these flags are discussed in section 6.3.1 of the *ProDOS 8 Technical Reference Manual*.

SmartPort Devices

Many of Apple's ProDOS block devices follow the SmartPort firmware interface. Through SmartPort, you can further identify devices. Existing SmartPort devices include SCSI hard disks, 3.5" disk drives and CD-ROM drives, with many more possible device types.

If $Cn07 = $00, then the device is a SmartPort device, and you can then make a SmartPort call to get more information about the device, including a device type and subtype. The SmartPort entry point is three bytes beyond the ProDOS block device entry point, which you already determined. The method for making SmartPort calls is outlined in the *Apple IIc Technical Reference Manual, Second Edition* and the *Apple IIGS Firmware Reference*.

The most useful SmartPort call to make for device identification is the `STATUS` call with `statcode` = 3 for Return Device Information Block (DIB). This call returns the ASCII name of the device, a device type and subtype, as well as the size of the device. Some SmartPort device types and subtypes are listed in the referenced manuals, with a more complete list located in the *Apple IIGS Firmware Reference*. A list containing SmartPort device types only is provided in SmartPort Technical Note #4, SmartPort Device Types.

RAM-Based Drivers

One fork of the identification tree comes to an end at this point. If the high byte of the device driver entry point was not $Cn and the device was not /RAM, you assumed it was a Disk II and used the slot number portion of the unit number to examine the slot ROM space. If the ROM space for that slot number does not match the three ProDOS block device ID bytes, it cannot be a Disk II. Having ruled out other possibilities, it must be a device installed after ProDOS finished building its device table. Perhaps it is a third-party RAM disk driver or maybe a driver for an older card which does not match the ProDOS block device ID bytes.

Whatever the function of the driver, you can identify it no further. It quite literally could be any kind of device at all, and with neither slot ROM space to identify nor a standard location to compare the device driver entry point against, the best you can do is consider it a "generic device" and go on.

But Is It Connected and Can I Read From It?

Just because a ProDOS device is in the table does not mean it is ready to be used. There is always the possibility that the drive has no media in it. Back in the beginning, you made an `ON_LINE` call with a unit number of $00. If the volume name of a disk in that device could not be read, or another error occurred, ProDOS 8 would return the error code in the `ON_LINE` buffer immediately following the unit number. Those errors possible include:

$27	I/O error
$28	No Device Connected
$2B	Write Protected
$2F	Device off-line
$45	Volume directory not found
$52	Not a ProDOS disk
$55	Volume Control Block full
$56	Bad buffer address
$57	Duplicate volume on-line

Note that error $2F is not listed in the *ProDOS 8 Technical Reference Manual*.

By convention, you interpret I/O error to mean the disk in the drive is either damaged or blank (not formatted). You interpret Device off-line to mean that there is no disk in the drive. You interpret No Device Connected to mean the drive really does not exist (for example, asking for status on a second Disk II when only one is connected).

If no error occurred for a unit number in the `ON_LINE` call (the low nibble of the unit number

is not zero), the volume name of the disk in the drive follows the unit number.

Things To Avoid

The ProDOS device-level `STATUS` call generally returns the number of blocks on a device. Applications should **not** try to identify 3.5" drives by doing a ProDOS or SmartPort `STATUS` call and comparing the number of blocks to 800 or 1,600. The correct way to identify a 3.5" drive is by the `Type` field in a SmartPort `STATUS` call.

Don't assume the characteristics of a device just because it is in a certain slot. For example, be prepared to deal with 5.25" disk drives in slots other than 6. Don't assume that slot 6 is associated with block devices at all—there could be a printer card installed.

Avoid reinstalling /RAM when your application finds it removed. If you remove /RAM, you should reinstall it when you're done with the extra memory; however, if your application finds /RAM already gone, you do **not** have the right to just reinstall it. A driver of some kind may be installed in auxiliary memory, and arbitrary reinstallation of /RAM could bring the system down.

Further Reference

- *ProDOS 8 Technical Reference Manual*
- *AppleShare Programmer's Guide for the Apple IIGS* (APDA)
- ProDOS 8 Technical Note #15, How ProDOS 8 Treats Slot 3
- ProDOS 8 Technical Note #20, Mirrored Devices and SmartPort
- ProDOS 8 Technical Note #23, ProDOS 8 Changes and Minutia
- ProDOS 8 Technical Note #26, Polite Use of Auxiliary Memory