AppleTalk
#1: Identifying AppleTalk

Revised by: Jim Luther March 1990
Written by: Dan Strnad November 1988

This Technical Note describes the correct methods for identifying AppleTalk under ProDOS 8 and GS/OS, as the `ATLK` ROM signature is no longer used.
Changes since July 1989: Added warning concerning ProDOS 8, version 1.4.

To determine if an application has been launched over the network, refer to the `NetLaunch` code fragment found in the *AppleShare Programmer's Guide for the Apple IIGS*.

Under ProDOS, to identify both AppleTalk and the slot with which it is associated for printing, refer to Apple II AppleTalk Technical Note #4, Printing Through the Firmware.

To identify AppleTalk under ProDOS 8:

1. Issue an AppleShare `GetInfo` call.
2. If there is no error result, AppleTalk is installed.

```
InfoParams    DB $00                    ;Synchronous only
              DB $02                    ;GetInfo call number
InfoResult    DS 13                     ;<- results returned here

CheckATalk    JSR $BF00
              DB  $42                   ;$42  command  #  for  AppleTalk
calls
              DW InfoParams             ;Parameter list address
              BCS NoATalk               ;handle the error
IsATalk.      ...                       ;AppleTalk installed when here

NoATalk       ...                       ;AppleTalk  not  installed  when
here
```

Warning: Due to a bug in ProDOS 8, version 1.4, using the $42 call crashes ProDOS 8 if AppleTalk is not installed. Applications that use this routine to check for AppleTalk should ship with ProDOS 8 version 1.5 or greater, thus avoiding this bug. (ProDOS 8 Technical Note #21, Identifying ProDOS Devices contains a routine which correctly identifies the presence AppleTalk under all versions of ProDOS 8.)

To identify AppleTalk protocols and AppleShare file system under System Software 5.0:

1. Set up the parameter block for a GS/OS `GetFSTInfo` call using `fstNum` = 1.
2. Issue the `GetFSTInfo` call.
3. If the `fileSysID` is $0D the AppleShare FST and AppleShare are present.
4. If a parameter out of range error ($53) results, the AppleShare file system is not present.
5. Otherwise, if steps 3 and 4 are inconclusive, increment the `fstNum` and loop back to step 2.

To identify AppleTalk protocols, including LAP through PFI but excluding the file system, under System Software 5.0:

1. Set up the parameter block for a GS/OS `DInfo` call using device number one.
2. Issue the `DInfo` call.
3. If the `deviceID` is $1D, the AppleTalk main driver and AppleTalk are present.
4. If a parameter out of range error ($53) results, the AppleTalk protocols are not present.
5. Otherwise, if steps 3 and 4 are inconclusive, increment the device number and loop back to step 2.

To identify AppleTalk protocols, including LAP through ASP but excluding the file system, under System Software 4.0:

1. Issue an an `SPGetStatus` call
2. If the call returns without error, AppleTalk is present.

Note: With the release of System Software 5.0, earlier versions are not supported.

Further Reference

- *Inside AppleTalk*
- *AppleShare Programmer's Guide for the Apple IIGS*
- *GS/OS Reference*
- Apple II AppleTalk Technical Note #4, Printing Through the Firmware
- ProDOS 8 Technical Note #21, Identifying ProDOS Devices

AppleTalk
#2: ProDOS 8 Compatibility on the IIe and IIGS

Written by: Mark Day

November 1988

This Technical Note describes areas which could cause an application to run under the AppleShare Apple IIe workstation software, but fail under the Apple IIGS workstation software.

- If code is running in auxiliary memory in emulation mode (e.g., ProDOS 8 programs that run code from auxiliary memory), make sure $0100 in auxiliary memory is set to the normal stack pointer and $0101 in auxiliary memory is set to the auxiliary (alternate) stack pointer. (See page 93 of the *Apple IIe Technical Reference Manual*.)
- Make sure ProDOS calls are not made from auxiliary memory; Apple has never recommended doing this, and it is not supported.
- Make sure interrupts are enabled when making ProDOS 8 calls.
- Make sure interrupts are not disabled for long periods of time, nor for a high percentage of time. Whenever interrupts are disabled, there is a chance that an AppleTalk packet will be missed (which could cause AppleShare volumes to be unmounted). The more interrupts are disabled, the more likely that packets will be missed. This risk is inherent for any application that disables interrupts (directly or indirectly), therefore, interrupts should be disabled with discretion and only when absolutely necessary.
- Make sure programs get the completion routine return address from the `GetInfo` call when they are started.
- Make sure to identify AppleTalk by calling `GetInfo` and checking for an invalid call number error (which means AppleTalk is not present). Do not use the `ATLK` signature bytes for identification. See Apple II AppleTalk Technical Note #1, Identifying AppleTalk.

Further Reference

- *Apple IIe Technical Reference Manual*
- Apple II AppleTalk Technical Note #1, Identifying AppleTalk

Apple II
Technical Notes

Developer Technical Support

AppleTalk
#3: Avoiding Remote Printer Time-Outs

Revised by: Jim Luther September 1989
Written by: Jim Luther May 1989

This Technical Note discusses how to avoid time-outs when printing to remote printers.
Changes since May 1989: Updated to reflect System Software 5.0 changes and to clarify the results of changing the time-out interval.

The Apple II AppleTalk firmware's Remote Print Manager (RPM), which supports AppleTalk's Super Serial Card (SSC) entry points, maintains a time-out interval value. The time-out interval is usually set to 30 seconds. When an application quits writing to the AppleTalk firmware, the RPM waits this time interval before sending the last block of data to the printer and closing the Printer Access Protocol (PAP) connection.

What does this mean? If an application waits longer than the time-out interval (e.g., 30 seconds) between any write accesses to the AppleTalk firmware (i.e., a pause between initialization and printing or a pause during printing), the PAP connection closes, the current page may be ejected from the printer (this is printer dependent—the ImageWriter II and ImageWriter LQ do not automatically eject the page, the Apple LaserWriter does), and the rest of the application's output to the printer is lost. If you initialize the AppleTalk SSC firmware, you must print immediately or a time-out may occur and reinitialization is necessary to print again. Applications should not initialize the firmware and expect it still to be initialized at a later point in time.

What You Can Do

The RPM's `PMSetPrinter` call may be used to change the time-out interval to a different value. However, the time-out interval should be kept as short as possible because other users cannot open another PAP connection with the printer until your machine has timed-out. In other words, if you set the time-out interval for five minutes, the RPM keeps the PAP connection open with the printer for five minutes after the last character is written to the RPM, thus blocking other machines from using that printer for five extra minutes; this delay is unacceptable in a shared printer environment.

With an Apple IIGS using System Software 5.0, the RPM's `PMSetPrinter` call may be used to set the time-out interval to zero. When the time-out interval is set to zero, the session never times out and must be closed with the Apple IIGS-specific `PMCloseSession` RPM call.

Further Reference

- *AppleShare Programmer's Guide for the Apple IIGS*

AppleTalk
#4: Printing Through the Firmware

Revised by: Jim Luther — September 1990
Written by: Matt Deatherage & Jim Luther — July 1989

This Technical Note discusses considerations of printing through the AppleTalk Remote Print Manager (RPM) interface from ProDOS 8 applications.
Changes since March 1990: Revised code sample to simplify finding the transparent network printing slot with the ROM 03 Apple IIGS. Please note that the method of finding the transparent network printing slot shown in the March 1990 revision of this Note *does* work correctly, the new method is just simpler. In addition, revised the wording of the Note to clarify that transparent network printing is the RPM interface.

The *AppleShare Programmer's Guide to the Apple IIGS* stated that the Remote Print Manager (RPM) interface allowed transparent network printing through Super Serial Card entry points in slot 7. This statement is pretty short-sighted. It's much like saying printing to an ImageWriter II is initiated when you do a `PR#1` command—it's only true if what you want is where you think it is—and usually it isn't.

Note: The *AppleShare Programmer's Guide to the Apple IIGS* has been superseded by the *AppleShare Programmer's Guide to the Apple II Family.*

An Apple IIe Workstation Card, although **recommended** for slot 7, can work in almost any slot (just like an ImageWriter II with a Super Serial Card can be connected to nearly any slot, except maybe slot 3 when the 80-column firmware is active). An Apple IIGS with ROM versions 00 or 01 may only have the firmware used by the RPM interface in slot 7. An Apple IIGS with ROM version 03 may only have the firmware used by the RPM interface in either slot 1 or 2.

Before printing through the RPM interface slot, take the same precautions you would take before printing to **any** slot—check to make sure you see the requested slot is a Pascal device before using Pascal entry points, and try to look for the signature bytes that indicate the features you want are present. In general, avoid hard-coding slot numbers for **anything**.

ProDOS 8 applications which offer network printing through the RPM interface should give users the choice of printing to any of the seven slots as well as the Network Printer. When Network Printer is selected, the application can find the slot used by the RPM interface by using the 6502 code sample included in this Note. Allowing the selection of Network Printer is especially important for applications that keep a configuration file containing a user's default printer setup. If an application keeps only the slot number in the configuration file, users may need to change the printer selection often if they print from several different machines.

Warning: Printing to a slot with no firmware generally results in a crash.

The code sample uses two methods to determine the slot the RPM interface is using. The first method works with the Apple IIe Workstation card and the ROM 01 Apple IIGS. It looks at the AppleTalk completion routine address returned by the AppleTalk `GetInfo` call, and if that address is in the slot ROM space, then that slot is the slot used by the RPM interface. In other words, if the completion routine points to $0000CnXX, where n is between 1 and 7, then n is the slot to be used when printing through the RPM interface. If the completion routine address is not in the slot ROM space, then the application cannot determine what slot the RPM interface is using and must query the user. The second method works only with the ROM 03 Apple IIGS. It retrieves the slot the RPM interface is using from location $E101C2.

This technique applies only to ProDOS 8 programs. Apple IIGS applications running under GS/OS should do text printing over the network through the Remote Print Manager (.RPM) driver, which can be identified by a `deviceID` of $001F as returned from the `DInfo` call.

```
;
; This routine will identify AppleTalk and the RPM interface slot (if possible).
; This routine is for ProDOS 8 applications only.
;
                    keep FindRPMSlot
                    longa off
                    longi off

FindRPMSlot         start
                    lda #$00
                    sta RPMSlot         default to no RPM interface slot

; Check for AppleTalk (see AppleTalk Technical Note #1)

                    jsr $BF00           ProDOS 8 MLI
                    dc  h'42'           $42 command for network calls
                    dc  a'InfoParams'   Parameter list address
                    bcs NoATalk         no AppleTalk; handle the error

; Get machine type & ROM version (see Apple II Miscellaneous Technical Note #7)

                    sec
                    jsr $FE1F           What kind of machine are we on?
                    bcs CheckCom        Not a IIGS, check completion address
                    cpy #$03
                    bcc CheckCom        Earlier than ROM 03 IIGS, check
;                                         completion address

ROM03               anop                ROM 03 or greater IIGS' use location $E101C2
;                                          to find the RPM interface slot
                    lda $E101C2         get the RPM interface slot
                    sta RPMSlot

                    beq AskForSlot
                    bra HaveSlot

CheckCom            anop                use completion address to find slot
                    lda ComReturn+2     bank $00?
                    ora ComReturn+3     high byte = 0?
```

```
bne AskForSlot      no, so slot can't be determined
lda ComReturn+1     get the address page
cmp #$C8
bcs AskForSlot      greater or equal to $C8 is bad
cmp #$C1
bcc AskForSlot      less than $C1 is bad
and #$0F            $Cn = $0n
sta RPMSlot
```

```
HaveSlot            anop                AppleTalk is installed and RPM is using
;                                         slot #RPMSlot

AskForSlot          anop                AppleTalk is installed but RPM interface slot
;                                         cannot be determined

NoATalk             anop                AppleTalk is not installed

                    rts                 so this sample returns

RPMSlot             entry
                    dc  h'00'           Slot RPM interface is using

InfoParams          dc  h'00'           Synchronous only
                    dc  h'02'           GetInfo call number
                    ds  2               result code
ComReturn           ds  4               completion return address
                    ds  8               space for other result info

                    end
```

Further Reference

- *AppleShare Programmer's Guide for the Apple II Family*
- Apple II AppleTalk Technical Note #1, Identifying AppleTalk
- Apple II Miscellaneous Technical Note #7, Apple II Family Identification
- Apple II Miscellaneous Technical Note #8, Pascal 1.1 Identification Bytes

AppleTalk
#5: SPCommand Calls and Error $0702

Written by: Mark Day July 1989

The system now uses `SPCommand` calls asynchronously. Applications that have AppleShare volumes mounted under System Software 5.0 and also make `SPCommand` calls themselves should now handle the “Too many ASP calls” error, $0702.

AppleShare uses a protocol called AppleTalk Session Protocol (ASP) to maintain a connection (session) with all servers that you are logged on to. All commands and data transfer to the server are sent using ASP.

The implementation of ASP on the Apple IIGS has a limit of one command outstanding (waiting to complete) per session. This means that if one command has been sent, its reply must be received before you can send the next command. Remember, the `SPCommand` call is used to send commands over a session. If you try to issue an `SPCommand` before another (asynchronous) `SPCommand` on the same session has completed, your call will return with a “Too many ASP calls” error, $0702.

Before System Software 5.0 on the Apple IIGS, no system software made asynchronous `SPCommand` calls, and therefore this error would only occur if the developer was making the asynchronous calls. As of System Software 5.0, the AppleShare FST uses asynchronous calls to help prevent the loss of a connection with servers and to assist the Finder in dynamically updating windows when a change is made to a network volume. Therefore, this error may be returned even though the developer is not making asynchronous calls.

The error is easy to handle if you are making synchronous `SPCommand` calls. Simply make the call, and if it completes with error $0702, loop back and make the call again until you can do so without error $0702. This technique forces your program to wait until ASP is free again to make the call.

If you are making asynchronous `SPCommand` calls, and you receive the $0702 error, you might want to install a short (i.e., 1/4 second) timer using the `InstallTimer` call, and make the `SPCommand` call again when the timer completes. Remember, the `InstallTimer` has to be asynchronous, since you are making it from the completion routine of an asynchronous call.

The `SPWrite` call also has a limit of one outstanding call per session. System software does not currently use asynchronous `SPWrite` calls, but looping until ASP returns something other than $0702 would be a good precaution for `SPWrite`, too.

Note: When using the AppleShare FST under GS/OS, there is little reason to make `SPCommand` calls yourself, since most of the calls you can make are available through the FST as normal file system calls or as FST-specific calls.

Further Reference

- *AppleShare Programmer's Guide for the Apple IIGS*
- *Inside AppleTalk*
- System Software 5.0 documentation (APDA)

AppleTalk
#6: Apple IIe Workstation Card Anomalies

Written by: Dan Strnad July 1989

This Technical Note describes known anomalies when using the Apple IIe Workstation Card.

- Pascal Protocol Serial `STATUS` call returns incorrect results.
  When using the Workstation card, the Pascal `STATUS` call (normally used for printing) does not properly indicate whether the card is ready to receive characters. Applications should avoid this call, as the Pascal `WRITE` call in the firmware will perform this function automatically.

- ProDOS 8 invisible bit is not respected.
  The invisible bit in the ProDOS 8 access byte was defined after the release of the Apple IIe Workstation Card, so the ProDOS Filing Interface present on the card treats this bit as reserved.

Further Reference

- *AppleShare Programmer's Guide for the Apple IIGS*
- *Apple IIe Technical Reference Manual*

AppleTalk
#7: MLIACTV Flag and the IIe Workstation Card

Written by: Mark Day & Dan Strnad November 1989

This Technical Note describes a problem using the `MLIACTV` flag with the IIe Workstation Card.

When using the Apple IIe Workstation Card, the `MLIACTV` flag does not always show that the MLI (or PFI) is active. This inconsistency can cause programs that use the `MLIACTV` flag to fail when making MLI calls from interrupt routines. Programs can correct for this problem by making all MLI calls through the `NewMLI` routine listed in this Note and checking the `NewMLIActv` flag instead of the `MLIACTV` flag. This approach solves the problem only if **all** MLI calls, including those made by any interrupt routines, are made through this routine.

The following routine is a replacement for the MLI entry point at $BF00. Programs using this routine can perform a `JSR` to `NewMLI` instead, which fixes the problem. Section 6.2.1 of the *ProDOS 8 Technical Reference Manual* details how programs can cause the MLI to return the their routine rather than the routine that originally called it. For programs using this technique that are also using the routine below, the location below labeled `NewCmdAddr` replaces `CmdAdr` ($BF9C). The steps involved in patching the MLI return location still apply, as specified in Section 6.2.1 of the ProDOS 8 Technical Reference.

```
; MLI patch for Apple II Workstation Card
; by Mark Day
;
; code shown is compatible with MPW IIGS cross-assembler
;
; Your program should use the NewMLIActv flag instead of
; MLIACTV ($BF9B), and should JSR NewMLI instead of
; JSR MLI ($BF00).
;

             machine      M6502        ; 6502 code for //e
             longa  off
             longi  off

parmptr      equ    0           ; two bytes on zero page
MLI          equ    $BF00       ; entry to the real MLI

NewMLI proc

             php                ; save old interrupt status to
             pla                ; temporarily disable interrupts
             sta oldp           ; so that NewCmdAddr is always valid
             sei                ; when an interrupting routine sees
                                ; NewMLI active.
             sec
             ror    NewMLIActv    ; NewMLI is now active!
```

```
;
; We need to get the return address from the stack so we can
; get the command number and parameter block address which
; follow the JSR NewMLI, and so we can save NewCmdAddr.
;
             clc
             pla                 ; get low byte of parm address - 1
             sta    parmptr
             adc    #4           ; get real return address
             sta    NewCmdAddr
             pla
             sta    parmptr+1    ; save high byte of parm address - 1
             adc    #0
             sta    NewCmdAddr+1 ; save real return address

             lda oldp
             pha
             plp                 ; reinstate old interrupt status
;
; Now, we copy the call number and parameter list pointer that followed
; the JSR NewMLI, and copy them after a JSR to the real MLI.
;
             tya                 ; save Y on stack
             pha
             ldy    #1           ; offset to command number
             lda    (parmptr),y  ; get command number
             sta    NewCmdNum
             iny                 ; point to parm list ptr (low)
             lda    (parmptr),y
             sta    NewParmPtr
             iny
             lda    (parmptr),y
             sta    NewParmPtr+1
             pla                 ; unstack value of y register
             tay

;
; Now, call the real MLI with the user's command and parameter list
; and jump back to our caller.
;
             jsr    MLI          ; call the real MLI
NewCmdNum    dc.b   0            ; command number
NewParmPtr   dc.w   0            ; parameter list pointer

             php                 ; save C because LSR changes it!
             lsr    NewMLIActv   ; MLI is no longer active
             plp                 ; restore C
             dc.b   $4C          ; JMP absolute instruction
NewCmdAddr   dc.w   0            ; target of jump, caller's return address

NewMLIActv   dc.b   0            ; $80 bit set if MLI active
oldp         ds.b   1            ; used to preserve processor status

             endp
             end
```

Note that this routine also works on the Apple IIGS, even though the problem with the `MLIACTV` flag only affects Apple IIe Workstation Cards.

Further Reference

- *AppleShare Programmer's Guide for the Apple IIGS*
- *ProDOS 8 Technical Reference Manual*

AppleTalk
#8: Using The @ Prefix

Written by: Jim Luther & Dan Strnad September 1990

This Technical Note discusses use the @ prefix with multiple users.

Apple II computers on AppleShare networks feature a unique folder for each user on the server volume, called the user's folder. Server volumes containing these user's folders are called user volumes. User's folders exist on user volumes so that the system and applications have a standard place to store user-specific data on the network. All network volumes an Apple II can boot from are user volumes.

Under GS/OS, the @ prefix allows applications to automatically work with the user's folder. If a user launches your application from a server volume with a user volume mounted, GS/OS sets the @ prefix to the user's folder; otherwise it sets it to the application folder. The @ prefix can reduce design and coding effort for multilaunch features by providing the application with the system's best guess at where user-specific information should be stored. To safely use the user's folder feature, programmers need only remember to use the @ prefix with the GS/OS class 1 `Open` call (`requestAccess` = 1, 2, or 3). Using the @ prefix with the class 1 `Open` provides safe access to the file for as long as it remains open, without requiring any network-specific code.

Using the @ prefix is appropriate for applications that want to avoid network-specific programming while being reasonably well-behaved in a network environment. For example, applications may store printer defaults in the @ directory or use it as a default when prompting the user to choose a directory.

There are situations writing data to a file in the @ directory could result in other users overwriting the data; however, applications may reasonably require that users not allow these situations to occur In Table 1, the third through fifth cases are all situations in which this problem could occur. For best results, when opening a file for writing with the @ prefix, use access privileges that deny write access to other users. The GS/OS class one `Open` call always does this when `requestAccess` is non-zero. Without this precaution of denying write access to other users, the user's data is not protected from being overwritten while it is in use.

Application launched from…	Is a user volume present?	User name	@ prefix set to…	Is this case detectable?
local	maybe	any name	application prefix	yes
net	yes	(not guest)	user folder	yes
net	no	any name	application prefix	yes
net	yes	guest	guest folder	yes
net	yes	same as another user	user folder	special programming required

Table 1–Possible @ Prefix Configurations

Consider the third case. Although the application was launched from a server, the server does not contain a user's folder, nor is there any other mounted server containing a user's folder. In this case, if multiple users launch a single copy of the application from the same folder at the same time, each user's @ prefix would point to the same folder from which they all had launched the application. By denying other users write access when opening the file, you prevent users from overwriting each other's data. However, the other users are no longer prevented from overwriting the data when the user with write access closes the file, at which point a different user can write to the file. Therefore, using access privileges in combination with the @ prefix deters one user's data from being overwritten by another, but only so long as the file remains opened by the user with write access. This approach may provide sufficient protection for saving certain user configuration and preference information.

When saving work the user plans to resume later, this approach may not provide sufficient protection. In this situation, conflicts can also arise if the @ prefix is set to the application prefix rather than to the user's folder. It is up to the programmer to determine whether to use the @ prefix, how to use it, and whether this level of protection is sufficient for the particular data involved.

In addition to using the @ prefix (or the user path to which it attempts to refer) with access that denies other users permission to write to the file, applications can check to see if the user path could point to the same folder for multiple users at the same time. To do this, the application first checks to see if it was launched across the network. This is the case when class one `GetFileInfo` on the user path returns `fileSysID` = $0D. If the application was launched across the network, the user path could be set the same for multiple users if the user has logged on as a guest (`UserInfo` returns a null `userName`, the fourth case in Table 1) or if you are using the @ prefix and the system has set it to the application prefix on a non-user network volume (error $60 from `GetUserPath`, the third case in Table 1). If the application determines that the user prefix may be set the same for multiple users, then it could use an alternate approach to determine where the data is to be stored, by prompting the user for example.

Although it would be comparatively difficult for an application to determine whether multiple users with the same name were running the application from the same server (the fifth case in

Table 1), the documentation for the application could warn against this. The system does not provide any specific information about when this condition occurs.

One More Caution

One other caution to observe when using the @ prefix: since other applications are also storing data in the same user's folder, each application should be careful to reference distinct files. Regardless of how the application chooses to do this—by checking that the file being created does not already exist, by choosing a distinct name for the file, or by some other method—it should usually operate only on files of its own creation.

Programmers should keep in mind that the @ prefix is provided as a programming convenience. The AppleShare FST also provides the `GetUserPath` and `UserInfo` calls. In combination with `GetFileInfo`, these calls allow programmers to devise other, more customized approaches for determining where to save the user's data.

Further Reference

- *AppleShare Programmer's Guide for the Apple II Family*
- *GS/OS Reference*
- GS/OS Technical Note #10, How Applications Find Their Files

Developer Technical Support

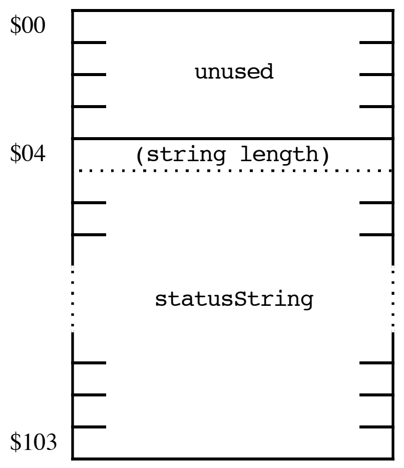
AppleTalk
#9: The PAP Status Buffer

Written by: Jim Luther November 1990

This Technical Note shows the format of the status data returned into the application-supplied status buffer by the `PAPStatus` and `PAPOpen` Printer Access Protocol (PAP) AppleTalk commands. The status buffer format is shown for both LaserWriter and ImageWriter (with the ImageWriter II/LQ LocalTalk Option card installed) printers.

The `PAPStatus` and `PAPOpen` AppleTalk commands must supply a pointer to a 260-byte status buffer. When the `PAPStatus` or `PAPOpen` commands complete, the status buffer contains the ATP data portion of a Status (TResp) packet. The first four bytes of that data are unused, so the actual status data starts at offset $04 in the status buffer.

The LaserWriter printer returns its status data in the form of a Pascal string. That string is usually something suitable to display on the screen (e.g., "status: idle" or "job: Fred; document: My LaserWriter is on fire; status: busy; source: AppleTalk"). In fact, the status text displayed in the Print Manager LaserWriter dialog boxes is usually the `statusString` returned by `PAPStatus` or `PAPOpen`. Figure 1 shows the contents of the status buffer returned by a LaserWriter.



Longword	Unused	
String	The PAP status string (Pascal string, ASCII, high-bit clear)	

Figure 1–PAP Status Buffer from a LaserWriter

The ImageWriter II/LQ LocalTalk Option card does not return a status string for display. Instead, it returns a `statusBits` word where each bit within that word has a specific meaning. Your application can interpret the `statusBits` word and generate an appropriate message to display. Figure 2 shows the contents of the status buffer returned by the ImageWriter II/LQ LocalTalk Option card and the individual bit definitions of the `statusBits` word.

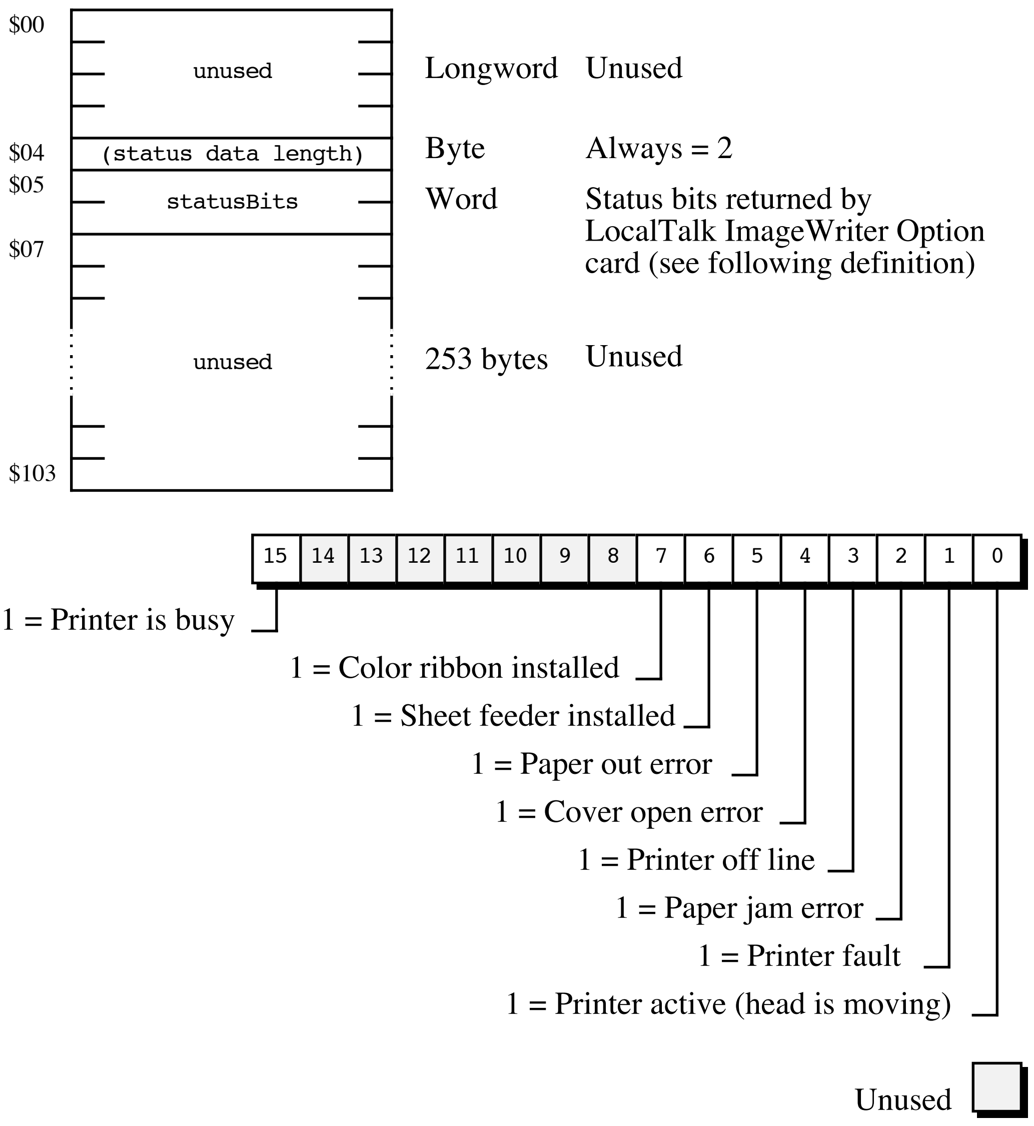


Figure 2–PAP Status Buffer from an ImageWriter II/LQ LocalTalk Option Card

There are two additional things to note when interpreting the `statusBits` word returned by a ImageWriter II/LQ LocalTalk Option card:

- If a sheet feeder is installed (bit 6 = 1), running out of paper results in a “Paper jam error” (bit 2 = 1) instead of a “Paper out error” (bit 5).
- The ImageWriter II/LQ LocalTalk Option card has been known to randomly return all ones in the low byte (bits 0-7) of the `statusBits` word. When this happens, the `statusBits` word is invalid and an application should repeat the `PAPStatus` call to get valid information.

Further Reference

- *Inside AppleTalk*, Second Edition
- *AppleShare Programmer’s Guide for the Apple II Family*