Technical Note IMERRATA03

Inside Macintosh: Networking Errata

CONTENTS

Topics

Chapter 2 - About the AppleTalk Utilities

Chapter 6 - AppleTalk Transaction Protocol (ATP)

Chapter 7 - Datagram Delivery Protocol (DDP)

Chapter 11 - Summary of Ethernet, TokenRing, and FDDI

References

Downloadables

This Technical Note discusses known errors and omissions in *Inside Macintosh: Networking* .

[Sep 01 1994]

Topics

- Clarification to the use of the `GetBridgeAddress` function. Sept 94
- Corrected special considerations to the use of `PSetSelfSend`.
- Socket Listener sample code correction/modification. Sept 94
- "Corrected" `EParamMisc2` C interface declaration is incorrect, Chapter 11 Sept 94
- ATP `transID` transaction ID field omitted Sept 94

Back to top

Chapter 2 - About the AppleTalk Utilities

Clarification to the use of GetBridgeAddress

Page 2-6, Getting the Address of Your Node or Your Local Router

The documentation states that "to get the node ID part of a local router's address, you can call the `GetBridgeAddress` function." This statement is not correct. Instead of returning the actual node ID of the router, the `GetBridgeAddress` function returns a non-zero function result if a router exists on the network. A function result of zero indicates that there is no router. To get the node ID of the router, use the `PGetAppleTalkInfo` function instead.

Correction to the Description for PSetSelfSend

Page 2-16, `PSetSelfSend` description.

The documentation states that "Sending packets between a multinode application and user node applications on the same machine is independent of the intranode delivery feature. A multinode is treated as a virtual node distinct from the user node...". These statements lead to the incorrect conclusion that one does not need to set the `SelfSend` capability to send packets between the user node and the multinode. To send packets between the user node and a multinode, use PSetSelfSend to turn on intranode delivery service.

Back to top

Chapter 6 - AppleTalk Transaction Protocol (ATP)

Transaction ID field transID omitted

The `transID` field should be used in place of the `reqTID` field as shown in the following areas:

Page 6-15

"The request transaction ID `transID` that ATP assigns to this request. If you intend to respond to the request, save this value because you will need to pass it to the `PSendResponse` function and the `PAddResponse` function to identify the request for which the response message is intended..."

Page 6-16

"For the input address block (`addrBlock`) and transaction ID (`transID`) parameters to `PSendResponse`, use the address block (`addrBlock`) and request transaction ID (`transID`) parameter values that the `PGetRequest` function returned."

Page 6-33

`PGetRequest` parameter block and description should look as follows:

Parameter block

-> `ioCompletion ProcPtr` A pointer to a completion routine.

<- `ioResult OSErr` The function result.

<- `userData LongInt` Four bytes of user data.

-> csCode Integer Always getRequest for this function.

-> atpSocket Byte The socket number.

<- atpFlags Byte The control information.

<- addrBlock LongInt The destination socket address.

<-> reqLength Word On input, the request buffer size. On return, the actual of the request received.

-> reqPointer Ptr A pointer to the request buffer.

<- bitMap Byte A bitmap.

<- transID Word The transaction ID.

Field descriptions

transID The transaction ID of the request that PGetRequest has received. ATP supplies this value.

"The PGetRequest function returns the transaction ID of the request that it receives in the transID field. You should save this value if you intend to respond to the request; this transaction ID is used as an input parameter to the PSendResponse and PAddResponse functions. To determine that the request transaction ID specified in the transID field is valid, first check the atpTIDValidvalue bit (bit 1) of the atpFlags field. If this bit is set, the transID field value is valid."

Page 6-37

Field descriptions

transID The transaction ID of the request for which this response is meant.

Page 6-43

Field should be an Integer, not a Byte:

-> transID Integer The transaction ID of the request with which the PSendResponse function to be canceled is associated.

Page 6-48

Add the transID field to the GetRequestParm parameter block as follows:

GetRequestParm:

(bitMap: Byte; {bitmap}

filler1: Byte;

transID: Integer);

Page 6-57

Add the transID field to the GetRequest Parameter Variant as follows:

GetRequest Parameter Variant

22 reqTID word request transaction ID

26 csCode word command code; always getRequest

29 atpFlags byte control information

30 addrBlock long destination socket address

34 reqLength word request size in bytes

36 reqPointer long pointer to request data

44 bitMap byte current bitmap

46 transID word request transaction ID

Back to top

Chapter 7 - Datagram Delivery Protocol (DDP)

Listing 7-6 Socket Listener sample code correction/modification.

Page 7-26 through 7-30, Receiving and processing a DDP packet

The sample code is provided to demonstrate a generic socket listener written in 68000 Assembler. The supplied code in this chapter does not correctly process a packet which is received with a checksum. A BRA.S instruction bypasses the portion of code which checks the packet for a checksum. In addition, the code for processing the checksum was not included in this release of *Inside Macintosh: Networking* . There is one other correction relating to the GetNextBuffer code. Before calling DeQueue, we must check for a nil pointer as DeQueue in some releases of System Software does not do this for us. The following is the complete socket listener code sample, including the SL_DoChksum code. Corrections to the supplied code for the socket listener only are presented in bold typeface.

In addition, the code has been modified to compile as a code resource which can be called using a Universal ProcPtr and executed in mixed mode, to facilitate compilation as native Power Macintosh code. The beginning of the code resource is the socket listener entry point, which is a JMP instruction to the actual listener code. The initialization code for the listener is two bytes into the code resource. For an example use of this listener code, refer to the Network Watch (DMZ) v1.3 application which is available on the Developer CD (Tool Chest Edition), August 1994 or later.

```
    INCLUDE    'QuickEqu.a'
    INCLUDE    'ToolEqu.a'
    INCLUDE    'SysEqu.a'
    INCLUDE    'ATalkEqu.a'
    INCLUDE    'Traps.a'
    INCLUDE    'SysErr.a'

;
;
; Record Types
;
;_____________________________________________________________________

MyQHdr                RECORD    0
qFlags                DS.W      1
qHead                 DS.L      1
qTail                 DS.L      1
                      ENDR

PacketBuffer          RECORD    0
qLink                 DS.L      1
qType                 DS.W      1
buffer_Type           DS.W      1           ; DDP Type
buffer_NodeID         DS.W      1           ; Destination node
buffer_Address        DS.L      1           ; Source address in AddrBlock format
buffer_Hops           DS.W      1           ; Hop count
buffer_ActCount       DS.W      1           ; length of DDP datagram
buffer_CheckSum       DS.W      1           ; Chksum error returned here
                                            ; (cksumErr or noErr)
buffer_Ticks          DS.L      1           ; TickCount when handler called
buffer_Data           DS.B      ddpMaxData  ; the DDP datagram
                      ENDR
```

```
THE_LISTENER    PROC    EXPORT
    BRA.S       TheListener
    BRA.S       SL_InitSktListener          ; branch to init code
;_____________________________________________________________________
;
; Local Variables
;
;_____________________________________________________________________

free_queue        DC.L    0      ; pointer to freeQ QHdr - init'd by InitSktListener
used_queue        DC.L    0      ; pointer to usedQ QHdr - init'd by InitSktListener
current_qelem     DC.L    0      ; pointer to current PacketBuffer record
                                 ; initialized by InitSktListener, then
                                 ; set by socket listener after every packet.
                                 ; NIL if no buffer is available.

;_____________________________________________________________________
; Function SL_InitSktListener(freeQ, usedQ: QHdrPtr): OSErr
;

StackFrame     RECORD    {A6Link},DECR     ; build a stack frame record
Result1        DS.W      1                 ; function's result returned to caller
ParamBegin     EQU       *                 ; start parameters after this point
freeQ          DS.L      1                 ; freeQ parameter
usedQ          DS.L      1                 ; usedQ parameter
ParamSize      EQU       ParamBegin-*      ; size of all the passed parameters
RetAddr        DS.L      1                 ; placeholder for return address
A6Link         DS.L      1                 ; placeholder for A6 link
LocalSize      EQU       *                 ; size of all the local variables
               ENDR

SL_InitSktListener:


    WITH       StackFrame,MyQHdr           ; use these record types

    LINK       A6,#LocalSize               ; allocate our local stack frame

; copy queue header pointers into our local storage for use in the listener

    LEA        used_queue,A0               ; copy usedQ into used_queue
    MOVE.L     usedQ(A6),(A0)

    LEA        free_queue,A0               ; copy freeQ into free_queue
    MOVE.L     freeQ(A6),(A0)

; dequeue the first buffer record from freeQ and set current_qelem to it

    MOVEA.L    freeQ(A6),A1                ; A1 = ^freeQ
    LEA        current_qelem, A0           ; copy freeQ.qHead into
                                           ; current_qelem
```

```
    MOVE.L     qHead(A1),(A0)
    MOVEA.L    qHead(A1),A0               ; A0 = freeQ.qHead
    _Dequeue
    MOVE.W     D0,Result1(A6)             ; Return status

@1  UNLK       A6                         ; destroy the link
    MOVEA.L    (SP)+,A0                   ; pull off the return address
    ADDA.L     #ParamSize,SP              ; strip all of the caller's
                                          ; parameters
    JMP        (A0)                       ; return to the caller

;_____________________________________________________________________
;
; SL_TheListner - process packets received at the designated socket
;
;    Input:
;     D0 (byte) = packet's destination socket number
;     D1 (word) = number of bytes left to read in packet
;     A0 points to the bytes to checksum
;     A1 points to the bytes to checksum
;     A2 points to MPP's local variables
;     A3 points to next free byte in Read Header Area
;     A4 points to ReadPacket and ReadRest jump table
;
;    Return:
;     D0 is modified
;     D3 (word) = accumulated checksum
;_____________________________________________________________________


    IMPORT  SL_DoChksum    : code

TheListener:

    WITH    PacketBuffer

; get pointer to current PacketBuffer

GetBuffer:
    LEA        current_qelem,A3          ; get the pointer to the PacketBuffer to use
    MOVE.L     (A3),A3
    MOVE.L     A3,D0                     ; if no PacketBuffer
    BEQ.S      NoBuffer                  ; then ignore packet

; read rest of packet into PacketBuffer.datagramData

    MOVE.L     D1,D3                     ; read rest of packet
    LEA        buffer_data(A3),A3        ; A3 = ^bufferData
    JSR        2(A4)                     ; ReadRest
    BEQ.S      ProcessPacket             ; If no error, continue
    BRA        RcvRTS                    ; there was an error, so ignore packet

; No buffer; ignore the packet

NoBuffer       CLR D3                    ; Set to ignore packet (buffer size = 0)
    JSR        2(A4)                     ; ReadRest
    BRA        GetNextBuffer             ; We missed this packet, but maybe there
                                         ; will be a buffer for the next packet...

; Process the packet you just read in.
; ReadRest has been called so registers A0-A3 and D0-D3 are free to use.
; We'll use registers this way:
PktBuff          EQU    A0        ; the current PacketBuffer
MPPLocals        EQU    A2        ; pointer to MPP's local variables (still set
                                  ;  up from entry to socket listener)
HopCount         EQU    D0        ; used to get the hop count
DatagramLength   EQU    D1        ; used to determine the datagram length
SourceNetAddr    EQU    D2        ; used to build the source network address

ProcessPacket:
    LEA        current_qelem,PktBuff          ; PktBuff = current_qelem
    MOVE.L     (PktBuff),PktBuff

; do everything that's common to both long and short DDP headers

; first, clear buffer_Type and buffer_NodeID to ensure their high bytes are 0

    CLR.W      buffer_Type(PktBuff)           ; clear buffer_Type
    CLR.W      buffer_NodeID(PktBuff)         ; clear buffer_NodeID

; clear SourceNetAddr to prepare to build network address

    MOVEQ      #0,SourceNetAddr               ; build the network address in
                                              ; SourceNetAddr

; get the hop count
```

```
    MOVE.W     toRHA+lapHdSz+ddpLength(MPPLocals),HopCount ; Get hop/length
                                                ; field
    ANDI.W     #DDPHopsMask,HopCount            ; Mask off the hop count bits
    LSR.W      #2,HopCount                      ; shift hop count into low bits of
                                                ; high byte
    LSR.W      #8,HopCount                      ; shift hop count into low byte
    MOVE.W     HopCount,buffer_Hops(PktBuff)    ; and move it into the
                                                ; PacketBuffer

; get the packet length (including the DDP header)
    MOVE.W     toRHA+lapHdSz+ddpLength(MPPLocals),DatagramLength ; Get length field
    ANDI.W     #ddpLenMask,DatagramLength       ; Mask off the hop count bits

; now, find out if the DDP header is long or short

    MOVE.B     toRHA+lapType(MPPLocals),D3      ; Get LAP type
    CMPI.B     #shortDDP,D3                     ; is this a long or short DDP
                                                ; header?
    BEQ.S      IsShortHdr                       ; skip if short DDP header

; it's a long DDP header

    MOVE.B     toRHA+lapHdSz+ddpType(MPPLocals),buffer_Type+1(PktBuff)
; get DDP type

    MOVE.B     toRHA+lapHdSz+ddpDstNode(MPPLocals),buffer_NodeID+1(PktBuff)
                                                ; get destination node from LAP
                                                ; header

    MOVE.L     toRHA+lapHdSz+ddpSrcNet(MPPLocals),SourceNetAddr
                                                ; source network in hi word
                                                ; source node in lo byte
    LSL.W      #8,SourceNetAddr                 ; shift source node up to high
                                                ; byte of low word
                                                ; get source socket from DDP
                                                ; header
    MOVE.B     toRHA+lapHdSz+ddpSrcSkt(MPPLocals),SourceNetAddr

    SUB.W      #ddpType+1,DatagramLength        ; DatagramLength = number of bytes
                                                ; in datagram

    ;BRA.S      MoveToBuffer <Delete this statement>

; Determine if there is a checksum
    TST.W      toRHA+lapHdSz+ddpChecksum(MPPLocals) ;Does packet have checksum?
    BEQ.S      noChecksum

; Calculate checksum over DDP header
    MOVE.W     DatagramLength,-(SP)       ; save DatagramLength (D1)

    CLR        D3                               ; set checksum to zero
    MOVEQ      #ddphSzLong-ddpDstNet,D1         ; D1 = length of header part to
                                                ; checksum pointer to dest network
                                                ; number in DDP header
    LEA        toRHA+lapHdSz+ddpDstNet(MPPLocals),A1
    JSR        SL_DoChksum                      ; checksum of DDP header part
                                                ; (D3 holds accumulated checksum)

; Calculate checksum over data portion (if any)

    LEA        buffer_Data(PktBuff),A1     ; pointer to datagram
    MOVE.W     (SP)+,DatagramLength        ; restore DatagramLength(D1)
    MOVE.W     DatagramLength,-(SP)        ; save DatagramLength (D1)
                                              ;  before calling SL_DoChksum
    BEQ.S      TestChecksum                     ; don't checksum datagram if its
                                                ; length = 0
    JSR        SL_DoChksum                      ; checksum of DDP datagram part
                                                ; (D3 holds accumulated checksum)

TestChecksum:
    MOVE.W     (SP)+,DatagramLength        ; restore DatagramLength(D1)

; Now make sure the checksum is OK.
    TST.W      D3                               ; is the calculated value zero?
    BNE.S      NotZero                          ; no -- go and use it
    SUBQ.W     #1,D3                            ; it is 0; make it -1

NotZero:
    CMP.W      toRHA+lapHdSz+ddpChecksum(MPPLocals),D3
    BNE.S      ChecksumErr                      ; Bad checksum
    MOVE.W     #0,buffer_CheckSum(A0)           ; no errors
    BRA.S      noChecksum
ChecksumErr:
    MOVE.W     #ckSumErr,buffer_CheckSum(PktBuff) ; checksum error

noChecksum:
```

```
    BRA.S      MoveToBuffer


; it's a short DDP header

IsShortHdr:
                                          ; get DDP type
    MOVE.B     toRHA+lapHdSz+sddpType(MPPLocals),buffer_Type+1(PktBuff)
                                          ; get destination node from LAP
                                          ; header
    MOVE.B     toRHA+lapDstAdr(MPPLocals),buffer_NodeID+1(PktBuff)
                                          ; get source node from LAP header
    MOVE.B     toRHA+lapSrcAdr(MPPLocals),SourceNetAddr
    LSL.W      #8,SourceNetAddr           ; shift src node up to high byte
                                          ; of low word

                                          ; get source socket from short DDP
                                          ;  header
    MOVE.B     toRHA+lapHdSz+sddpSrcSkt(MPPLocals),SourceNetAddr

                                          ; DatagramLength = number of bytes in
                                          ; datagram
    SUB.W      #sddpType+1,DatagramLength

MoveToBuffer:
                                    ;move source network address into PacketBuffer
    MOVE.L     SourceNetAddr,buffer_Address(PktBuff)
                                    ; move datagram length into PacketBuffer
    MOVE.W     DatagramLength,buffer_ActCount(PktBuff)

; Now that we're done with the PacketBuffer, enqueue it into the usedQ and get
; another buffer from the freeQ for the next packet.

    LEA        used_queue,A1                ; A1 = ^used_queue
    MOVE.L     (A1),A1                      ; A1 = used_queue (pointer to usedQ)
    _Enqueue                                 ; put the PacketBuffer in the usedQ

GetNextBuffer:
    LEA        free_queue,A1                ; A1 = ^free_queue
    MOVE.L     (A1),A1                      ; A1 = free_queue (pointer to freeQ)
    LEA        current_qelem, A0            ; copy freeQ.qHead into current_qelem
    MOVE.L     qHead(A1),(A0)
    MOVEA.L    qHead(A1),A0                 ; A0 = freeQ.qHead
    MOVE.L    A0,D0            ; check whether there is a queue element
    BEQ.S     RcvRTS           ; branch if not - don't dequeue nil ptr.
    _Dequeue

RcvRTS:
    RTS                                    ; return to caller

ENDWITH
    ENDP

;_____________________________________________________________________________
;
; SL_DoChksum - accumulate ongoing checksum (from Inside Macintosh)
;
;    Input:
;     D1 (word) = number of bytes to checksum
;     D3 (word) = current checksum
;     A1 points to the bytes to checksum
;
;    Return:
;     D0 is modified
;     D3 (word) = accumulated checksum
;_____________________________________________________________________________

SL_DoChksum    PROC
    CLR.W      D0                  ; Clear high byte
    SUBQ.W     #1,D1               ; Decrement count for DBRA
ChksumLoop:
    MOVE.B     (A1)+,D0            ; read a byte into D0
    ADD.W      D0,D3               ; accumulate checksum
    ROL.W      #1,D3               ; rotate left one bit
    DBRA       D1,ChksumLoop       ; loop if more bytes
    RTS
    ENDP
    END
```

Listing 7-7 sample code has logic error

Page 7-32, Testing for Available Packets

The sample code "Determining if the socket listener has processed a packet", incorrectly uses the following statement to check whether a packet was successfully Dequeued:

```
IF (Dequeue(QElemPtr(bufPtr),  @usedQ) <> noErr) THEN
```

The corrected statement is:

```
IF (Dequeue(QElemPtr(bufPtr),  @usedQ) = noErr) THEN
```

Back to top

Chapter 11 - Summary of Ethernet, TokenRing, and FDDI

"Corrected" EParamMisc2 C interface declaration is incorrect

Page 11-46, A corrected definition for the `EParamMisc2` variable type is presented as:

```
typedef struct {
    EParamHeader
    char     eMultiAddr[5];
} EParamMisc2;
```

This declaration is incorrect. The `eMultiAddr` field is 6 bytes long. The correct structure is defined as:

```
typedef struct {
    EParamHeader
    char     eMultiAddr[6];
} EParamMisc2;
```

Back to top

References

Inside Macintosh: Networking

Back to top

Downloadables

Acrobat version of this Note (56K) Download

Back to top