New Technical Notes

Macintosh

Developer Support

Link Access Protocol Q&As

Networking

Revised by: Developer Support Center May 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&As this month:
AppleTalk and VBL task using self-send mode

AppleTalk and VBL task using self-send mode
Date Written: 5/18/92
Last reviewed: 3/1/93

To make all AppleTalk calls asynchronously, I have to split the VBL task in pieces:

- Setselfsend
- check its async finish
- StartLookup
- check its async finish
- Restoreselfsend
- check its async finish

This means that while the selfsend is set to true, other software will be confused when they do a lookup. Also, when the Macintosh is doing I/O on the network, the VBL will set and restore the selfsend flag. Will other AppleTalk calls fail if the SelfSend flag Is set to true, or does selfsend only work on PLookupName?

—

Your method of splitting up the steps of a lookup to yourself during a VBL task using self-send mode is correct, except you needn’t bother restoring the state of self-send since you are giving up time to other processes between calls to your VBL task.

Self-send mode is a global setting and may affect any AppleTalk calls. When the LAP manager is preparing to send a packet, it checks the destination node. If the destination node is its own node address or the broadcast address, then it sends the packet to itself before putting the packet out on the wire. This usually only affects NBP lookup packets because they are sent to the broadcast address. Most other AppleTalk protocols do not broadcast.

Programs using self-send should follow these guidelines:

- If self-send is only needed for a brief period of time, then you should turn self-send on with PSetSelfSend (saving the current setting returned in oldSelfFlag), make the call(s) that require self-send, and then restore self-send to its previous state.

- If self-send is needed for an extended period of time (for example, for the life of your application) where your program will give up time to other programs, then you should turn self-send on and leave it on (don't restore it to its previous state). Since other programs running on your system (that aren't well behaved) may turn off self-send at any time, programs that require self-send mode should periodically call PSetSelfSend to make sure self-send stays on. Apple's system software has no compatibility problems with self-send —that is, it doesn't care if it's on or off—so leaving it on won't hurt anything.

How to tell if LAP Manager is present
Date Written: 12/12/91
Last reviewed: 2/3/92

What's the recommended technique for telling whether the user has turned off AppleTalk?

—

The best way to determine whether AppleTalk has been turned off is to use the AppleTalk Transition Queue to alert you to .MPP closures. (This is one of the reasons why the AppleTalk Transition Queue was implemented.) The AppleTalk Transition Queue is available only in AppleTalk version 53 or later, and is documented in the AppleTalk chapter of *Inside Macintosh* Volume VI, starting on page 32-17. There's also a code snippet, Transition Queue, in the Snippets folder on the *Developer CD Series* disc.

AppleTalk AOpen and AClose codes 7 & 8
Date Written: 11/1/91
Last reviewed: 11/27/91

What are AppleTalk messages 7 and 8? They're documented in lapequ.a as:

```
AOpen      EQU         7       ; Open an ATlk
AClose     EQU         8       ; Close an ATlk
```

but what should the adev do with them and what parameters are passed?

—

The AOpen and AClose messages are sent when the .MPP driver is opened and closed, respectively. When the AppleTalk driver is closed, a shutdown message is not sent. Instead, the LAP Manager determines whether an adev is version 3 or greater and, if true, assumes that

the adev can respond to the AClose message and sends the message. Conversely, when the .MPP driver is re-opened while the adev is active, the AOpen message is sent to the adev (assuming that the adev is version 3 or greater).

An important note on this matter is that if you modify your adev to version 3, additional selectors must be supported. These new selectors will be documented in a forthcoming Tech Note.

Use transition queue to make sure AppleTalk isn't turned off

Date Written: 10/8/91
Last reviewed: 10/8/91

How do I check if the user has turned off AppleTalk after I have already started using it?

—

The best way to determine this information is to submit an AppleTalk transition queue element so that whenever changes to the active state of AppleTalk occur, your program is notified. Check out *Inside Macintosh* Volume VI, page 32-17, the Macintosh Technical Note "AppleTalk Phase 2 on the Macintosh," which discuss this topic. A simple description of the process is as follows.

First, define a transition handler function, which has the following form:

```
/*
*  AppleTalk Transition Queue Sample
*
*    Macintosh Developer Technical Support
*    ©1990 Apple Computer, Inc.
*
*    10/31/90 pvh
*/

#include <AppleTalk.h>

/*
* Transition queue routines are designed with C calling conventions in mind.
* They are passed parameters with a C style stack and return values are
* expected to be in register D0.
*/

/* defined in IM VI, pgs 32-21&22, and MPW AppleTalk.h and AppleTalk.a
Includes*/
#define ATOpenEvent           0        /* MPP just opened */
#define ATCloseEvent          2        /* MPP is closing */
#define ATClosePrepEvent      3        /* OK for .MPP to close? */
#define ATCancelCloseEvent    4        /* MPP close was cancelled */

long main(long selector, ATQEntry *q, void *p)
{
   long    returnVal = 0;
   /*
    * This is the dispatch part of the routine. We'll check the selector
    * passed into the task; its location is 4 bytes off the stack (selector).
    */
   switch(selector) {
       case ATOpenEvent:
           /*
```

```
            * Someone has opened the .MPP driver. This is where one would
            * reset the application to its usual network state (i.e., you
            * could register your NBP name here). Always return 0.
            */
           break;
       case ATCloseEvent:
           /*
            * When this routine is called .MPP is going to shut down no
            * matter what we do. Handle that type of situation here (i.e.,
            * one could remove an NBP name and close down all sessions)
            * p will be nil. Return 0 to indicate no error.
            */
           break;
       case ATClosePrepEvent: {
           /*
            * This event gives us the chance to deny the closing of AppleTalk
            * if we want. Returning a value of 0 means it's OK to close,
            * non-zero indicates we'd rather not close at this time.
            * With this event, the parameter "p" actually means something.
            * p in this event is a pointer to an address which can hold a
            * pointer to a string of our choosing. This string indicates to
            * the user which task would rather not close. If you don't want
            * AppleTalk to close, but you don't have a name to stick in
            * there, you MUST place a nil value in there instead.
            * (We're doing this all locally to this case because it's C and
            * we can, so there.)
            */
           char **t;
           /*
            * Get a new reference to the address we were passed (in a form we
            * can use)
            */
           t = (char **) p;
           /*
            * Place the address of our string into the address we were passed
            */
           *t = "\pBanana Mail";  /* this will either be an Ax reference
                                     or PC relative, depending on compiler
                                     and options */
           /*
            * Return a nonzero value so AppleTalk knows we'd rather not
            * close.
            */
           returnVal = 1;
           break;
           }
       case ATCancelCloseEvent:
           /*
            * Just kidding, we didn't really want to cancel that AppleTalk
            * closing after all. Reset all you network activites that you
            * have disabled here (if any). In our case, we'll just fall
            * through. p will be nil.
            */
           break;
       default:
           /*
            * For future transition queue events (and yes, Virginia, there
            * will be more)
            */
           break;
   } /* end of switch */

   /*
```

```
    *    return value in register D0
    */
   return returnVal;
}
```

OK, the next thing to do is to use the LAPAddATQ function, passing along the address of the handler, for insertion of the handler into the transition queue. See *Inside Macintosh* Volume VI, page 32-18, and the referenced Tech Note for details. After that, your handler will be called every time some transition event takes place, even if it's one which you initiated—"like closing down AppleTalk," for example.

When the application is finished, remember to call LAPRmvATQ to remove the ATQ element; otherwise, AppleTalk will continue to send trans queue messages to a code resource which has since gone away. Oops.

X-Ref:
Macintosh Technical Note "AppleTalk Phase 2 on the Macintosh"
Inside Macintosh Volume VI, Chapter 32, "AppleTalk Manager"