New Technical Notes

Macintosh

Developer Support

AppleTalk Overview Q&As

Networking

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&As this month:
Documentation for creating an AppleTalk-aware network device

Documentation for creating an AppleTalk-aware network device
Date Written: 1/12/93
Last reviewed: 4/1/93

Where can I find information for creating an AppleTalk-aware network device?

—

The *Inside AppleTalk* reference provides much of the protocol information required for your device to communicate with networked Macintosh systems. Inside AppleTalk is available from APDA (#C0078LL/B).

For writing hardware device drivers, the *Macintosh AppleTalk Connections Programmer's Guide* discusses information on writing ADEVs and 'atlk' and driver resources. This guide helps one to understand the workings of the Link Access Protocol Manager, and how packets are handled to make sure they go out through the right port.

Apple doesn't provide information about the hardware design of an AppleTalk-aware device.

In addition, information concerning the hardware layer of LocalTalk hasn't been made available other than that provided in *Inside AppleTalk*. The best source of information on this subject probably is from the Internet or some other N&C development community. For Ethernet compatible devices, some of the information that you'll need can be obtained from the

developer notes available from the vendors of Ethernet chips, such as National Semiconductor, Inc., manufacturer of the SONIC chip.

Another useful tool for developing such a product is a network sniffer, so that you can watch the packet exchange between devices, and compare the exchange with a similar AppleTalk device. This will help in determining whether your device has correctly implemented the protocol. You can probably find product information regarding the various sniffers available by looking on the Redgate icon on AppleLink or in your favorite product catalog.

Broadcasting to all internet nodes
Date Written: 9/22/92
Last reviewed: 11/1/92

I want to send a Broadcast to all the nodes on an internet, whether EtherTalk or LocalTalk, when my application comes up. I'm not sure which AppleTalk protocol to use. I tried using ALAP and DDP but couldn't get either to work.

—

Unfortunately, you can't broadcast to an entire AppleTalk internet. In order to limit network overload, broadcast packets are localized. On AppleTalk, they don't extend over zones and, on Ethernet, they don't cross bridges. What you might do, however, is something roughly like the following:

Server side:

1. Initialize your application by creating a socket in the "server" address range:

```
ioParam.ioNamePtr = "\p.MPP";
ioParam.ioMisc = (Ptr) 0x80000000;  /* Server flag */
PBOpenSync(&ioParam);
```

2. Create an ATP socket:

```
POpenATPSkt(&atpParam, 0); /* Synchronous */
gMySocket = atpParam.ATP.atpSocket;
```

3. Register your name. The NBP objStr should be the name of the system your application is running on. You can use GetString(-16413) to get the system name on System 7 (or -16096 to get the Chooser name). The NBP typeString should be a name unique to your application. Note: be sure to remove the name before your application terminates. If you get error -1027 (nbpDuplicate), a previous instance of your application terminated without removing the name. Just call PRemoveName and try again.

4. Your server should now post an ATPRead and wait for messages.

Client side:

The "client" is the part of your application that looks for servers. The algorithm is, roughly, as follows:

1. Use GetZoneList to get a complete list of AppleTalk zones. Make sure you handle the "no zones" case correctly.

2. For each zone, use PLookupName to get a list of servers with the NPB typeStr that the “server side” registered.

3. For each found entity, use NBPExtract to get the entity name and the addrBlock. The entity name gives you the “human readable” text that identifies the network zone and system name, while the addrBlock lets you send ATP messages to the other system.

The AppleTalk program interface is described in *Inside Macintosh* Volumes V and VI. Use these as your primary reference, referring to *Inside Macintosh* Volume II only when necessary.

Note that you have to figure on at least 15 seconds per zone. This means that, if you’re trying this algorithm on some huge network (like the Apple engineering network), your application will probably need over a half-hour to probe every zone. The reason for mentioning this is that some applications are using network-uniqueness as a form of user license enforcement. If you do this, it would be unfair to the honest user to stall the application for a half-hour to check for duplicates.

Another application use would be for systems that maintain a central server, such as for a database, that handles many clients. Here, you probably should ask the user for the zone:node of the server, remember this in your application’s preference file, and look for that server.

One other thing: In writing this type of program, the SetSelfSend capability is extremely helpful: that way, you can debug both server and client functions on one machine — indeed in one application. Even if you don’t normally use SetSelfSend, make sure your application handles it as it is a global state for all AppleTalk connections on the machine.

How to get the User and Computer names used by AppleTalk

Date Written: 11/27/91
Last reviewed: 11/27/91

I’d like to use the same names that the system uses to identify itself on the AppleTalk network in my program. Where can I find those names?

—

The names used by the system for network services are stored in two 'STR ' resources in the System file. Your program can retrieve those names with the Resource Manager’s GetString function.

Only one of the names is available in systems before System 7: the name set by the Chooser desk accessory. That name is stored in 'STR ' resource ID -16096. With System 7, the Sharing Setup control panel lets the user assign two names for network services: the Owner name and the Computer name.

The Owner name is the name stored in 'STR ' resource ID -16096; it identifies the user of the Macintosh. The Owner name is used by System 7 for two primary purposes: to identify the owner of the system when accessing the system remotely through System 7 file sharing or through the user identity dialog used by the PPC Toolbox (and Apple Event Manager), and to serve as the default user name when logging on to other file servers with the Chooser.

The Computer name (also known as the Flagship name) is the name stored in 'STR ' resource ID -16413; it identifies the Macintosh. The Computer name is the name used by system network services to identify themselves on the AppleTalk network. For example, if your system's Computer name is "PizzaBox," the PPC Toolbox registers the name "PizzaBox:PPCToolBox@*" when you start program linking, and file sharing registers the name "PizzaBox:AFPServer@*" when you start file sharing.

Which AppleTalk routines can & can't be called at interrupt time
Date Written: 6/19/91
Last reviewed: 10/9/91

Can any AppleTalk routines be called at interrupt time? Inside Macintosh says that DDPWrite and DDPRead can't be called from interrupts. If all higher-level AppleTalk protocols are based on DDP, it seems that they all would not work.

—

The AppleTalk routines you can't call at interrupt time are the original AppleTalk Pascal Interfaces listed in Inside Macintosh Volume II; these are also known as the "Alternate Interface" AppleTalk routines, or ABPasIntf.

The Alternate Interface routines cannot be called at interrupt time because they allocate the memory structures needed to make the equivalent assembly language AppleTalk call. For example, when the NBPLookup routine is called, it's passed a handle to an ABusRecord. NBPLookup then has to allocate an MPPParamBlock and move the parameters from the ABusRecord into the newly allocated MPPParamBlock. Then NBPLookup makes a LookupName call, passing it the MPPParamBlock. When LookupName completes, NBPLookup must move results into the ABusRecord and release the memory used by the MPPParamBlock. Since memory is allocated and released within the routine, it cannot be called at interrupt time.

With that out of the way, the calls you can make at interrupt time (with some restrictions listed below) are what Apple calls the "Preferred Interface" AppleTalk routines. Most of the Preferred Interface routines are listed on page 562 of Inside Macintosh Volume V. There are a few additional calls that were added after the publication of Inside Macintosh Volume V; they're documented in the AppleTalk chapter of Inside Macintosh Volume VI.

The Preferred Interface AppleTalk routines can be made at interrupt time as long as:

• You make them asynchronously with a completion routine (that is, the asynch parameter must be TRUE and you must provide a pointer to the completion routine in the ioCompletion field of the call's parameter block). Making a call asynchronously and polling ioResult immediately afterward within the same interrupt-time code (which is basically the same as making the call synchronously) is not the same as using a completion routine.

• They are not listed as routines that may move or purge memory. The Preferred Interface routines do not allocate or dispose of any memory, since they're just high-level ways to make the assembly language AppleTalk calls and are not built upon the old Alternate Interface routines.

```
X-Ref:
Macintosh Technical Note "Using the High-Level AppleTalk Routines"
Macintosh Technical Note "AppleTalk Interface Update"
```

Macintosh Technical Note "Avoid Use of Network Events"

AppleTalk and INITs

Date Written: 9/5/91
Last reviewed: 9/17/91

Does AppleTalk work the same with an INIT as with an application? I'm using only Name Binding Protocol (NBP) and Datagram Delivery Protocol (DDP) stuff.

—

Yes! Responder is an example of an INIT that utilizes the NBP to register a Macintosh on the network. NBP requires DDP, so it is also present. AppleTalk is loaded before any INITs, which makes it available to them.

Install AppleTalk with Installer instead of drag-installing

Date Written: 12/3/91
Last reviewed: 12/11/91

We have problems using AppleTalk version 56 on a Macintosh Plus under System 6. The problems go away when we drop AppleTalk version 52 into the System Folder. What's going on?

—

AppleTalk version 56 isn't working for you because you just copied the AppleTalk file into the System Folder of your Macintosh with the Finder (this is known as a "drag-install"). All versions of AppleTalk from version 53 up through the current version must be installed by an Installer script. If you drag-install those versions of AppleTalk, several very important system resources won't be copied into the System file and AppleTalk won't work.

Apple software products that require AppleTalk version 53 or greater always include the Installer and Installer script that copy all system resources needed correctly. Apple also supplies the source to an Installer script that you can use if you license AppleTalk to ship with your products. See the "AppleTalk Licensing Disk 3.2" folder on the latest Developer CD for a look at the Installer script code. As usual, you'll need to contact Software Licensing at AppleLink address SW.LICENSE if you want to ship AppleTalk with your products.

Preferred AppleTalk calls and AppleTalk version

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

Do I need AppleTalk version 48 to call AppleTalk routines using the preferred AppleTalk interface?

—

The preferred AppleTalk routines don't require AppleTalk version 48. You can make any of the AppleTalk calls listed in *Inside Macintosh* Volume II from a Macintosh Plus. To make .XPP driver calls listed in Inside Macintosh Volume V, you'll need to use AppleTalk version 48. To make the new AppleTalk calls listed in *Inside Macintosh* Volume VI (and the Macintosh Technical Note "AppleTalk Phase 2 on the Macintosh"), you'll need version 53 or later. To

make ADSP calls to the .DSP driver, you'll need to either install ADSP or use AppleTalk version 56 or later which includes the .DSP driver.

AppleTalk Peek and Poke tools
Date Written: 5/2/91
Last reviewed: 8/1/92

Is there an update for the Apple tool called Poke?

—

The following versions of Peek and Poke are current:

AppleTalk Peek 3.1
AppleTalk Peek 3.2d5
AppleTalk Poke 3.1

You'll find them on the latest *Developer CD Series* disc, in Tools & Apps:Networking & Communications:AppleTalk Tools.

Using AppleTalk self-send mode
Date Written: 4/20/92
Last reviewed: 7/13/92

What's the recommended method for allowing an AppleTalk node to send packets to itself using AppleTalk's self-send mode (intranode delivery), assuming customers are running various versions of AppleTalk? There used to be a control panel called SetSelfSend that would turn on AppleTalk self-send mode at startup time. Should we use that control panel or should we use the PSetSelfSend function in our program to set the self-send flag ourselves?

—

AppleTalk self-send mode requires AppleTalk version 48 or greater. You can check the AppleTalk version with Gestalt or SysEnvirons. All Macintosh models except for the Macintosh XL, 128, 512, and Plus have AppleTalk version 48 or greater in ROM.

The SetSelfSend control panel is still available on the Developer CD Series disc (Tools & Apps:Intriguing Inits/cdevs/DAs:Pete's hacks-Moof!:SetSelfSend). However, we don't recommend it as a solution if you need to use self-send mode in your program. Instead, you should use the PSetSelfSend function to turn self-send mode on with your program.

AppleTalk's self-send mode presents a problem. Any changes made to the state of self-send will affect all other programs that use AppleTalk. That is, self-send mode is global to the system. Because of this, programs using self-send should follow these guidelines:

• If you need self-send for only a brief period of time (for example, to perform a PLookupName on your own node), you should turn it on with PSetSelfSend (saving the current setting returned in oldSelfFlag), make the call(s) that require self-send, and restore self-send to its previous state.

• If you need self-send for an extended period of time (for example, the life of your application) in which your program will give up time to other programs, you should turn self-send on and

leave it on — do not 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 should periodically check to make sure it's still on with either PSetSelfSend or PGetAppleTalkInfo. 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.

Documentation describing AppleTalk 56 and 57 differences
Date Written: 8/14/92
Last reviewed: 10/11/92

What are the differences between AppleTalk 56 and 57 in terms of features, bug fixes, interface, and any technical changes related to driver interfaces?

—

The primary difference between the two versions of AppleTalk involves support for AppleTalk Remote Access, and multinodes. "Multinode" is the capability for a workstation to acquire a node ID separate from that of the workstation user node. AppleTalk Remote Access uses a multinode to provide services for the remote connection via the modem. For more information concerning the differences between the two revisions, there are two documents that you should read. The Macintosh Technical Note "AppleTalk, the Rest of the Story" (formerly #311, "What's New With AppleTalk Phase 2") explains many of the changes. You can also read the release notes, located on the Developer CD by this path:

Dev.CD Aug 92: Tools & Apps: Networking & Communications: AppleTalk Release Notes:
AppleTalk 56-->57 Changes

This document discusses many of the bug fixes implemented in the new release.

Different AppleTalk versions OK for nodes
Date Written: 8/14/92
Last reviewed: 10/11/92

Is it OK for different nodes to run different versions of AppleTalk on a network?

—

You don't need to run the same version of AppleTalk for each node on the network. However, AppleTalk versions 53 and greater support AppleTalk Phase 2, so if you're working on an application that depends on the Phase 2 support (for instance, using the NBP wildcard character "≈"), you'll need to use AppleTalk version 53 or later for all nodes running that application.