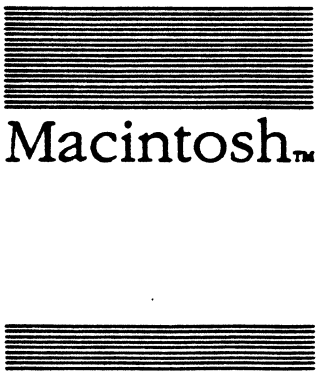


# Macintosh™ EtherTalk and Alternate AppleTalk Reference

Alpha Draft; Working Draft 1 - December 11, 1987

Communications & Networking
Apple Technical Publications

Engineering Part No. 6588279

CONFIDENTIAL

# APPLE COMPUTER, INC.

This manual is copyrighted by Apple, with all right reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given or lent to another person. Under the law, copying includes translation into another language.

© Apple Computer, Inc., 1987
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.
AppleTalk is a registered trademark of Apple Computer, Inc.

NuBus™ is a trademark of Texas Instruments.

Ethernet® is a registered trademark of Xerox.

Simultaneously published in the United States and Canada.

# Contents

# Figures and tables

# Preface

This preliminary note is intended to be used by Apple® software developers who wish to develop an alternate AppleTalk® implementation or Ethernet application in conjunction with the Macintosh™ operating system. To make use of the information presented here, you should have a working knowledge of the existing AppleTalk environment and, depending on your application, a working knowledge of Ethernet.

## Document Contents

This preliminary note provides you with an interactional overview of Apple's EtherTalk™ software, as well as a detailed description of each software component. Call definitions, register usage, and call applications are discussed.

## Suggested Reading

Here is a list of reference materials that relate or apply directly to the EtherTalk network environment:

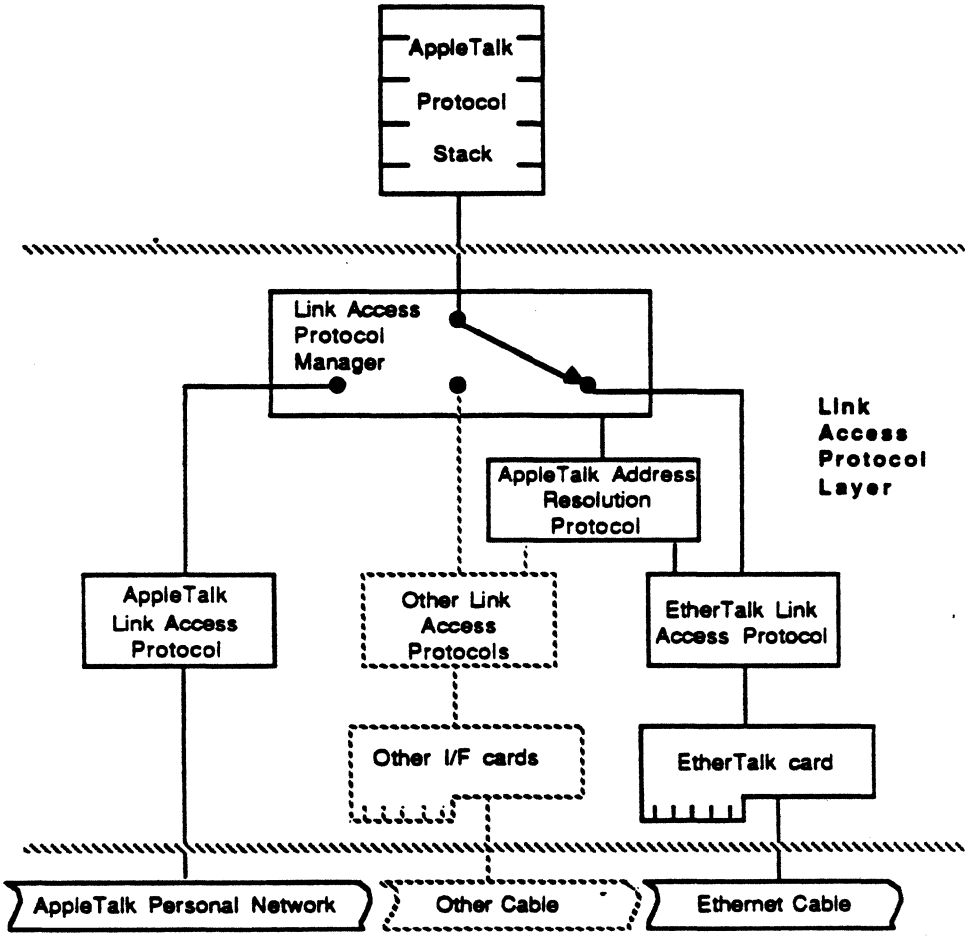
- Sidhu, Gursharan S., Richard F. Andrews and Alan B. Oppenheimer, *Inside AppleTalk* (Apple Programmers and Developers Association)
- *InInside Macintosh*, Volume II, Chapter 6 "The Device Manager" (Apple Computer Inc.)
- *Inside Macintosh*, Volume II, Chapter 10 "The AppleTalk Manager" (Apple Computer Inc.)
- *Inside Macintosh*, Volume V, Chapter 23 "The Device Manager" (Apple Computer Inc.)
- *Inside Macintosh*, Volume V, Chapter 24 "The Slot Manager" (Apple Computer Inc.)
- *Inside Macintosh*, Volume V, Chapter 28 "The AppleTalk Manager" (Apple Computer Inc.)
- *Ethernet Blue Book* (Xerox Inc.)
- *EtherTalk User Guide* (Apple Computer Inc.)

# Chapter 1

# Introduction

## AppleTalk

The name AppleTalk refers to a *system* of hardware and software components that transfer information when connected by a physical medium. The AppleTalk Personal Network (APN), EtherTalk, AppleShare™, and LaserShare™ are all components of the AppleTalk system. Figure 1-1 shows the AppleTalk system as it interacts with the APN and EtherTalk implementations on a Macintosh II computer:




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 1-1**
AppleTalk Implementations

Apple developed a specific set of rules, or communication protocols, to control the transfer of information among all nodes on the network. These AppleTalk protocols correspond to the various layers (Physical, Data Link) of the International Standard Organization-Open Systems Interconnection (ISO-OSI) reference model.

◆ *Note:* Refer to ***Inside AppleTalk*** and ***Inside Macintosh,*** Volume II for more information about AppleTalk protocols.

## AppleTalk Implementations

In addition to transferring information over the APN cabling scheme, AppleTalk protocols can now transfer information over a higher-performance AppleTalk connection— EtherTalk. EtherTalk, for the Macintosh II, consists of the EtherTalk interface card and a software package which enables transmission and reception of AppleTalk packets over Ethernet coaxial cable and allows compatibility with Ethernet.

Before the development of EtherTalk, the only option available to the user was to transfer information over the APN or equivalent cabling system by using the AppleTalk Link Access Protocol (ALAP) to perform node-to-node delivery of information. While this process was sufficient for many situations, the Macintosh could transfer information on only the APN. To expand the networking capability of the Macintosh, Apple chose to incorporate a Link Access Protocol (LAP) Manager to perform a "switching" function that can direct AppleTalk protocol information to the APN, Ethernet, or any other LAPs that support additional networks.

### LAP Functions

The ALAP assigns a unique identification number to each device, or node, on the APN. This identification number, known as the node ID, is an 8-bit address that ALAP dynamically assigns at node-startup time. The 8-bit node ID works well on the APN and is required by the AppleTalk protocols; however, the Ethernet data link only recognizes 48-bit addresses. The EtherTalk Link Access Protocol (ELAP) parallels the ALAP function of assigning addresses by using another protocol—AppleTalk Address Resolution Protocol (AARP). The EtherTalk implementation of AARP converts, or maps, a series of 8-bit AppleTalk node IDs and their 48-bit Ethernet equivalents. This Preliminary Note discusses AARP and the driver-level ELAP in more detail in later chapters.

## Using EtherTalk

EtherTalk software is designed to operate with the Macintosh operating system and, more specifically, the Macintosh II computer. The Macintosh II may contain as many as six EtherTalk interface cards to allow connection to multiple Ethernet cabling systems. Any Macintosh computer may operate EtherTalk software as long as a compatible Ethernet interface card and driver are present. A high-level look at EtherTalk software reveals new network icons as shown in Figure 1-2.

MSC NNNN
ART: NN x 8.5 pi
12 pi text to FN b/b

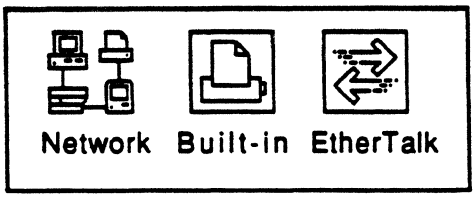



**Figure 1-2**
Network Icons

When the user selects the Network icon from the Control Panel, the content area of the Control Panel's window displays the icons for all AppleTalk connections supported by the system, of which EtherTalk is only one. The Built-in icon represents the AppleTalk Personal Network.

- *Note:* AppleTalk must still be active in the Chooser for any AppleTalk implementation to operate.

In addition to the Control Panel software, EtherTalk software also contains these components:

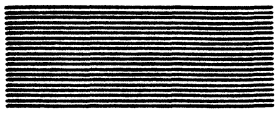
- the Ethernet Driver, which is the interface to the Ethernet card.
- the Link Access Protocol (LAP) manager, which standardizes interaction with AppleTalk drivers.
- the AppleTalk Address Resolution Protocol (AARP), which performs Ethernet-AppleTalk address mapping; and which may also perform address mapping between AppleTalk addresses and other networks.
- the LAP Manager INIT Resource, which informs the system of which AppleTalk connection to use at startup time.



CONFIDENTIAL

## Possible Applications

There are many possible applications that you may wish to develop. For example, you may want to create your own alternate AppleTalk implementation or to develop an Ethernet driver for use with a different interface card. Other applications might be to make Ethernet calls directly on a Macintosh, create your own AARP, or develop an EtherTalk implementation for use on another device.



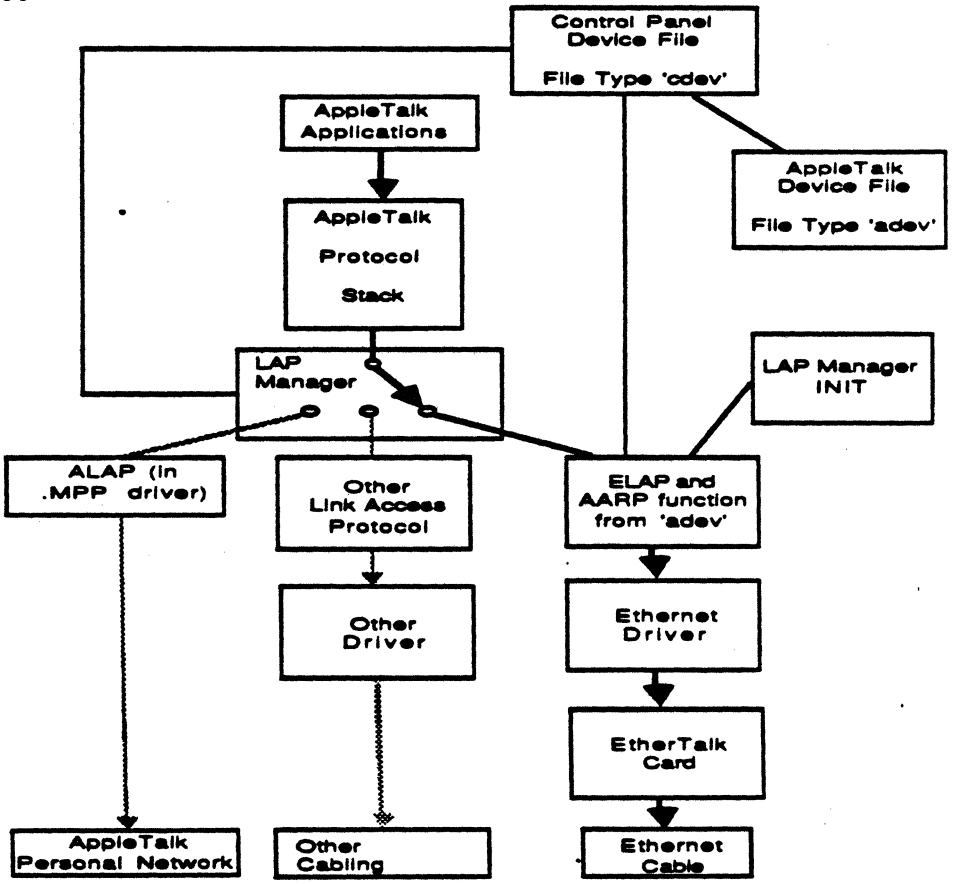
# Chapter 2

# EtherTalk Overview

This chapter identifies the contents of each component of EtherTalk software and discusses their interaction and, to some extent, their application. Later chapters discuss each piece of EtherTalk software in more detail.

## Block Diagram

Figure 2-1 shows all EtherTalk components and the way that these components relate to the AppleTalk environment.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-1**
EtherTalk Component Relationship

## Device Files 'adev' and 'cdev'

AppleTalk Device files (file type 'adev') and the Control Panel Device files (file type 'cdev') both reside in the System Folder. These device files work together to display a scrollable list of icons in the left side of the Control Panel. EtherTalk software contains the Network 'cdev' file which, when selected, displays a series of icons to represent each AppleTalk connection. Each alternate (other than Built-in) AppleTalk implementation must have its own 'adev' file.

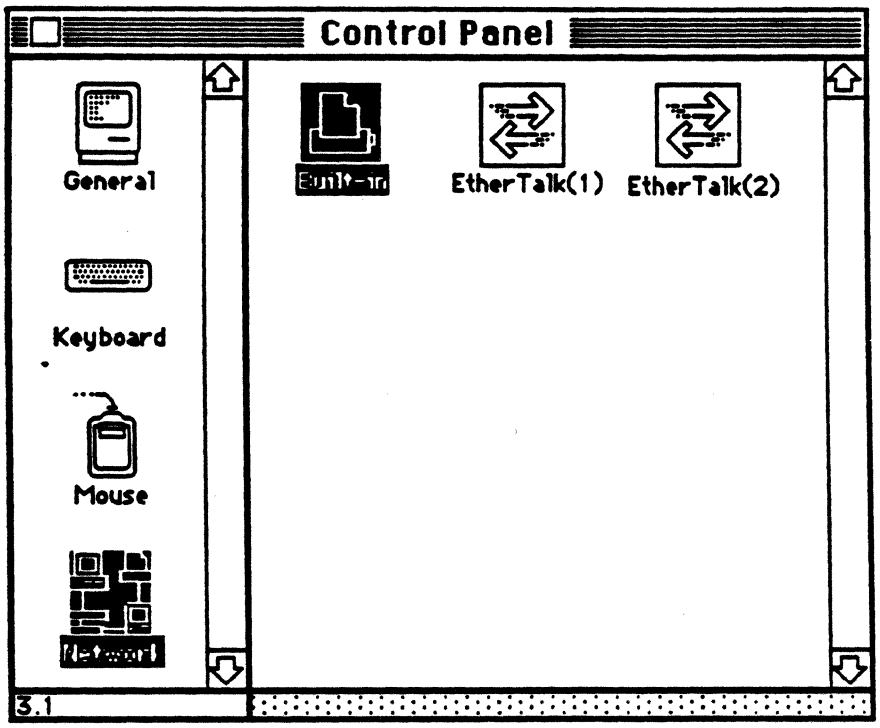
### Control Panel Device File

Control Panel Device files, which are of file type 'cdev', contain various resources that communicate machine options in some form (buttons, icons, and so on) to the user via the Control Panel. These 'cdev' files also handle user events such as clicks and keystrokes. Examples of 'cdev' files are the General, Mouse, Keyboard, and Color files.

EtherTalk software contains a new 'cdev' file called Network. The Network 'cdev' file, located in the System Folder, allows the user to select one AppleTalk connection from a list of others. The Network 'cdev' file contains various resources to display user-interface selections and to communicate selection information to the system.

When the user selects the Control Panel from the Apple Menu, the Control Panel scans the System Folder for files of type 'cdev'. Upon finding a 'cdev' file, the Control Panel takes the file's icon and title (string) and adds them to the scrollable list on the left side of the Control Panel. When all the icons are added to the scrollable list, the Control Panel selects the General icon and constructs the control information in the window's content area. At this point, the user may select any one of the icons in the scrollable list.

Figure 2-2 shows the Built-in AppleTalk selection and the alternate AppleTalk selections EtherTalk(1) and EtherTalk(2) that are available to the user after the Network icon is selected.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-2**
Control Panel

❖ *Note:* The term *Network 'cdev'* refers to the resources that comprise the Network 'cdev' file.

When the user selects the Network icon, the Network 'cdev' scans the System Folder for all files of type 'adev'. As the Network 'cdev' accesses each 'adev' file, each 'adev' file responds by passing various information back to the Network 'cdev' file, including the 'adev' icon and icon string. Each 'adev' file may support more than one AppleTalk connection of the same type and, if so, must also instruct the Network 'cdev' as to the number of identical icons to display and the strings for each.

For example, if two EtherTalk cards are installed in the Macintosh II, a single 'adev' file that supports both cards informs the Network 'cdev' to display two EtherTalk icons and place an identifying string under each icon. For EtherTalk, the Network 'cdev' places the string EtherTalk(n) under each icon, where 'n' equals each card's slot number.

After all icons appear in the content area of the Control Panel, the user may select one of the AppleTalk icons for use. When the user makes a new selection, the Network 'cdev' highlights this icon and performs various operations to inform the system of the new AppleTalk selection.

## AppleTalk Device File

The construction of an 'adev' file is similar to that of a 'cdev' file. For each alternate AppleTalk implementation such as EtherTalk, the 'adev' file must contain the following resources:

- 'ICN#'
- 'STR '
- 'BNDL'
- 'FREF'
- owner resource
- 'adev' code resource
- 'atlk' code resource

The 'ICN#" and 'STR ' resources are the icon and the string that the Network 'cdev' file displays in the Control Panel content area for the alternate AppleTalk implementation. In addition, if the 'adev' file contains the 'BNDL', 'FREF', and owner resources, and the 'adev' file has it's bundle bit set, the 'ICN#' will appear as the custom icon in the Finder.

## The 'adev' and 'atlk' Resources

The 'adev' and 'atlk' resources are pieces of stand-alone code. The 'adev' resource is responsible for handling all interaction with the Network 'cdev'. The Network 'cdev' loads the 'adev' resource into the application heap, calls the 'adev' resource to identify or to select an AppleTalk implementation, and removes the 'adev' resource as the Network 'cdev' requires.

The 'atlk' resource contains the actual implementation code for the alternate AppleTalk selection. The Network 'cdev' loads the 'atlk' resource into the system heap, calls for initialization and installation, and detaches the 'atlk' resource. Because the Network 'cdev' detaches the 'atlk' resource, the current alternate AppleTalk selection remains in effect when the 'adev' file closes.

## The LAP Manager INIT Resource

When the user makes an alternate AppleTalk selection from the Control Panel, the Network 'cdev' updates parameter RAM with a value that represents the current AppleTalk selection. This value remains in parameter RAM when the Macintosh is powered off.

At boot time, the LAP Manager INIT resource, located in the System File, interacts with the 'atlk' resource in much the same manner as the Network 'cdev' does. This INIT resource obtains the last AppleTalk selection value from parameter RAM, loads the corresponding 'atlk' file into the system heap, calls the 'atlk' resource for initialization, and then detaches the 'atlk' resource.

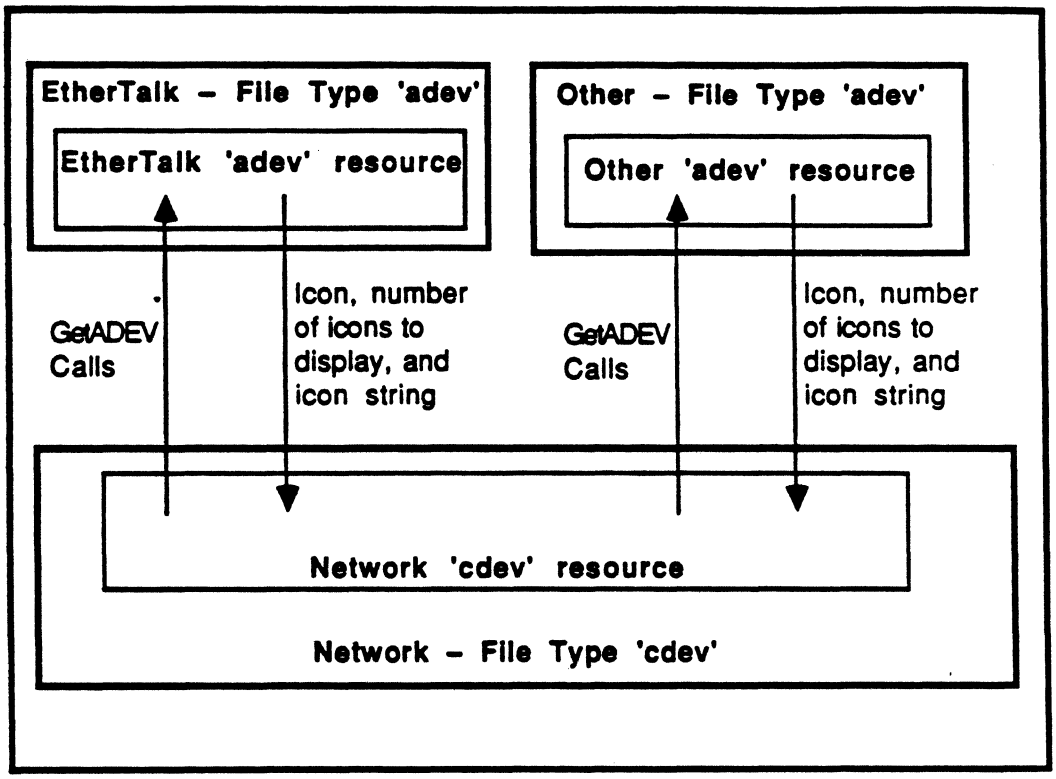
- *Note:* The LAP Manager INIT resource also loads the LAP Manager into memory and initializes the LAP Manager at startup time.

## Calls to the 'adev' Resource

When the user selects a new alternate AppleTalk implementation form the Control Panel, the Network 'cdev' makes two calls to the 'adev' resource to handle the user interface. These two calls are GetADEV and SelectADEV.

### The GetADEV Call

When the user selects the Network 'cdev' icon in the Control Panel, the Network 'cdev' makes a series of GetADEV calls to each 'adev' resource in the System Folder. Each 'adev' resource responds by telling the Network 'cdev' how many icons to display and by identifying the string ('STR ') for each icon. Figure 2-3 shows this interaction.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-3**
The GetADEV Call

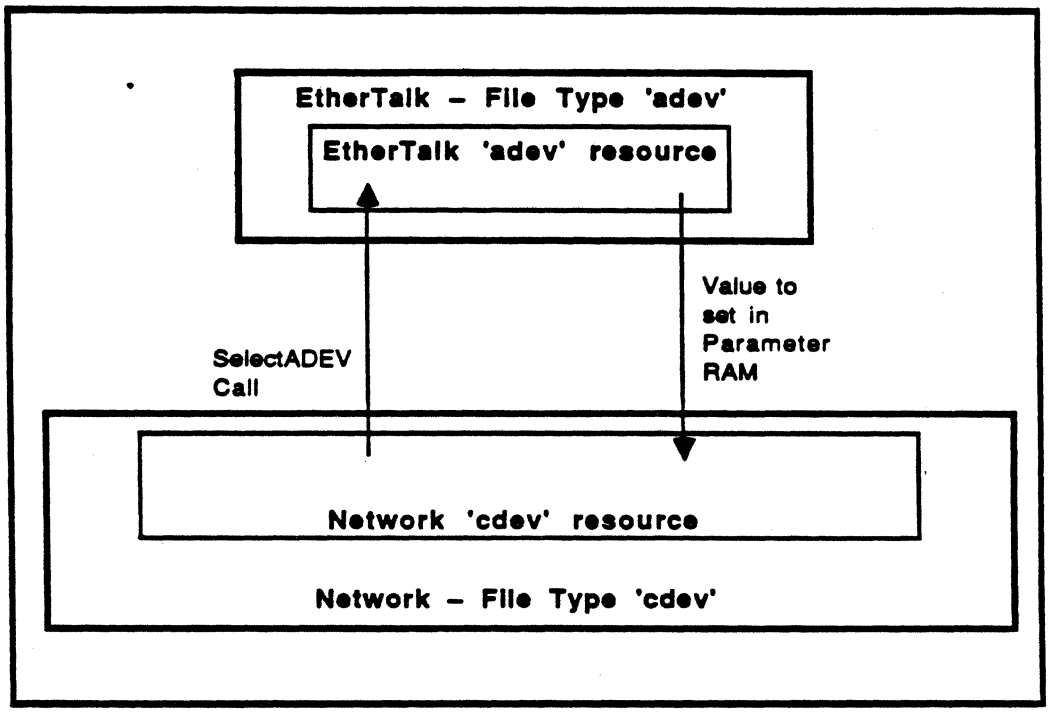
◆ *Note:* The Network 'cdev' does *not* make the GetADEV call to the Built-in AppleTalk code. The Built-in code is part of the Network 'cdev' file.

### The SelectADEV Call

When the user clicks on an alternate AppleTalk icon, the Network 'cdev' makes a SelectADEV call to the 'adev' resource to indicate the selection and determine the value that the AppleTalk selection wants to place in parameter RAM. This value indicates the details of the AppleTalk selection.

For example, imagine that a Macintosh II contains two EtherTalk cards and displays two EtherTalk icons. The user selects one icon. The Network 'cdev' makes a SelectADEV call to the 'adev' resource. The 'adev' resource returns a value to the Network 'cdev' to indicate which card is currently selected. This value is eventually passed to the 'atlk' resource and placed in parameter RAM.

Figure 2-4 illustrates the SelectADEV call for EtherTalk.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-4**
The SelectADEV Call

❖ *Note:* Refer to Chapter 3 for more information about calls to the 'adev' resource.

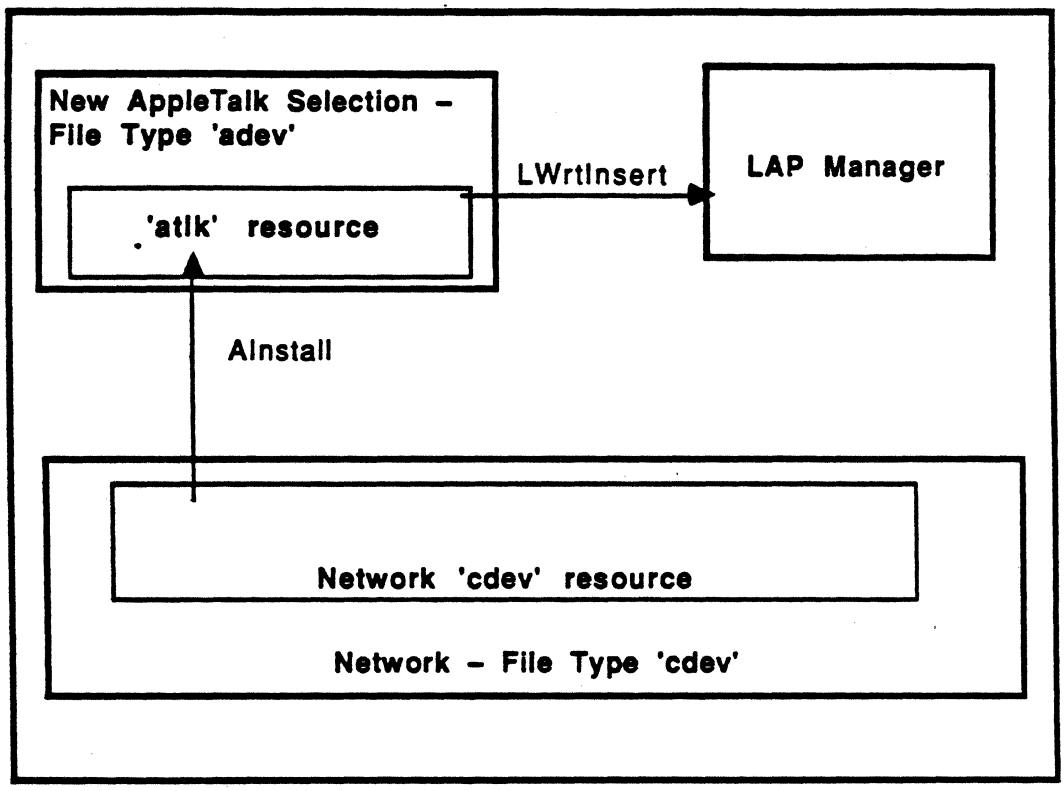
## Calls to the 'atlk' Resource

In addition to making the SelectADEV call, the Network 'cdev' must also close down the previously selected AppleTalk implementation and install the new user selection as the current AppleTalk implementation. To close down the AppleTalk selection, the Network 'cdev' makes an AppleTalk Shutdown (AShutdown) call to the 'atlk' resource. To install the new AppleTalk selection, the Network 'cdev' makes an AppleTalk Install (AInstall) call to the 'atlk' resource.

### The AInstall and LWrtInsert Calls

After the Network 'cdev' calls the 'adev' resource with the SelectADEV call, the Network 'cdev' loads the new 'atlk' code into the system heap, calls the 'atlk' resource with an AInstall call, and detaches the 'atlk' resource so that it remains in memory when the user closes the Control Panel.

Generally, in response to the AInstall call, the 'atlk' code makes a Lap Write Insert (LWrtInsert) call to the LAP Manager to tell the LAP Manager to install a portion of the 'atlk' code into the LAPWrite hook which is a low-memory location equal to ATalkHk2 . The portion of 'atlk' code that the LAP Manager loads into low memory is responsible for sending packets. At startup time, the LAP Manager INIT resource also loads the 'atlk' resource, as indicated by parameter RAM, into the system heap; calls with AInstall; and detaches it. Figure 2-5 illustrates the AInstall and LWrtInsert calls.



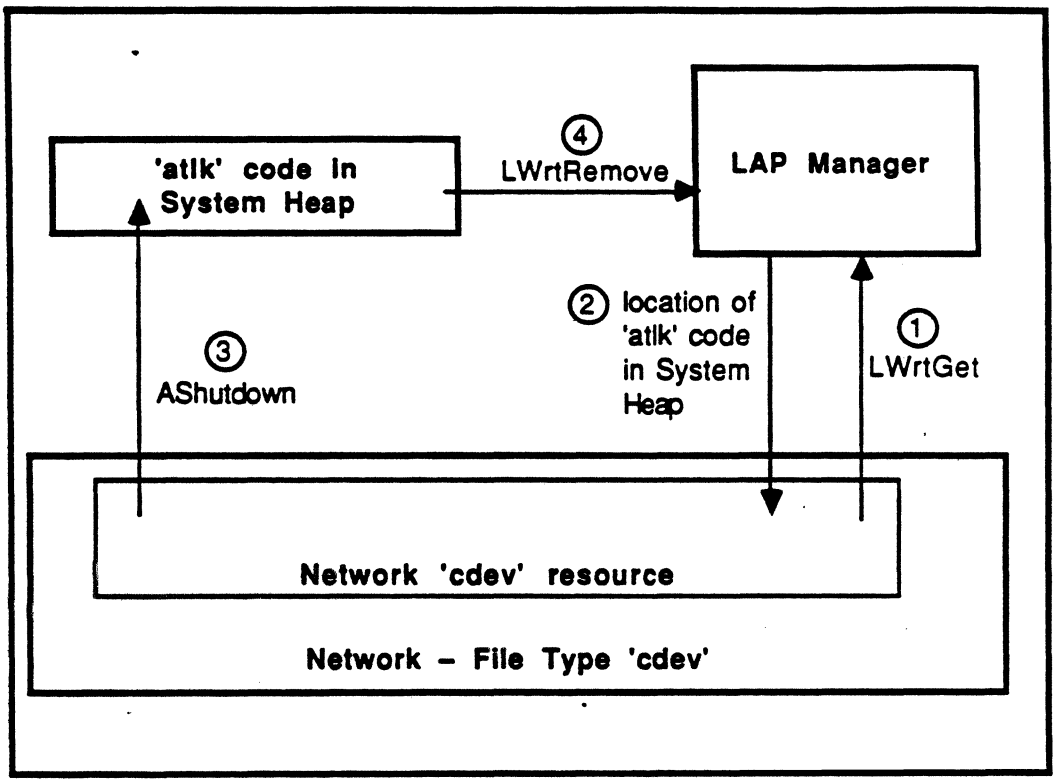

MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-5**
AInstall and LWrtInsert Calls

### The LWrtGet, AShutdown, and LWrtRemove Calls

To close down the previous AppleTalk selection, the Network 'cdev' makes an AShutdown Call to dispose of the 'atlk' resource. However, the Network 'cdev' has detached the previously installed 'atlk' code and does not know its location in the system heap. Before making the AShutdown call to the 'atlk' code, the Network 'cdev' makes a Lap Write Get (LWrtGet) call to the LAP Manager to obtain the location of the 'atlk' code. In general, after the Network 'cdev' makes the AShutdown call, the 'atlk' code should respond by making a LAP Write Remove (LWrtRemove) call to the LAP Manager. The LAP Manager responds to this call by removing the old 'atlk' code in the LAPWrite hook. Normally, the AShutdown and LWrtRemove calls are made before the AInstall and LWrtInsert calls.

Figure 2-6 illustrates the LWrtGet, AShutdown, and LWrtRemove calls.



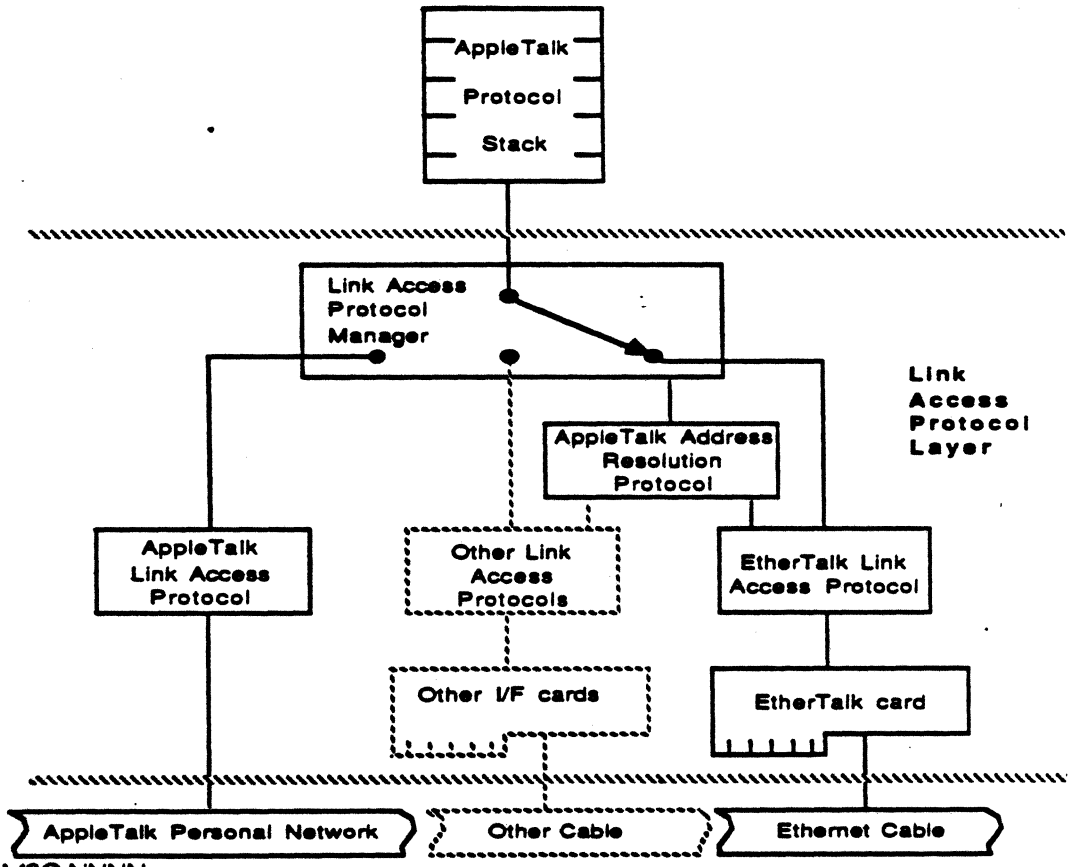

MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-6**
LWrtGet, AShutdown, and LWrtRemove Calls

## The LAP Manager

The LAP Manager standardizes interactions with the AppleTalk drivers/protocol stack and the link access protocol layer of the current AppleTalk selection. By standardizing these interactions, various AppleTalk implementations will not interfere with each other and will not have to make use of information that is private to the AppleTalk drivers.

The LAP Manager resides between the link access protocols of all AppleTalk implementations and the AppleTalk protocol stack, as Figure 2-7 shows.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 2-7**
LAP Manager Position

The LAP Manager is installed in the system heap at startup time, before the AppleTalk drivers are opened. The LAP Manager takes control of the LAPWrite hook, which is located in low memory as ATalkHk2. The AppleTalk drivers use the LAPWrite hook to direct outgoing AppleTalk packets.

## AppleTalk Selection

The LAPWrite hook contains the code that is, for all practical purposes, the actual AppleTalk implementation for outgoing packets. The LAP Manager installs this code in LAPWrite hook under the direction of the code itself. In the case of EtherTalk, the 'atlk' code resource tells the LAP Manager which portion of the 'atlk' code to insert in the LAPWrite hook. Loading 'atlk' code into the LAPWrite hook happens at two different times:

- whenever the user makes an AppleTalk selection from the Control Panel
- at startup time when the INIT resource obtains the AppleTalk selection value from parameter RAM.

### Installing the AppleTalk Selection

As indicated earlier in this chapter, when the user makes a new AppleTalk selection, the Network 'cdev' loads the 'atlk' code into the system heap. The 'atlk' code then makes a LWrtInsert call to the LAP Manager. The LWrtInsert call contains a pointer which tells the LAP Manager the location of the portion of the'atlk' code to insert into the LAPWrite hook as the AppleTalk selection.

### Intranode Delivery

The LAP Manager handles the sending of an AppleTalk packet to its own node unless the 'atlk' code specifies otherwise. If the LAP Manager is to handle intranode packets, the LAP Manager generally will *not* call the 'atlk' code for packet delivery. However, if the LAP Manager is to handle intranode delivery and an application sends a broadcast packet to the network, the LAP Manager will handle the intranode delivery of this packet and will call the 'atlk' code for packet transmission on the network.

## Packet Reception

When the 'atlk' code receives an incoming AppleTalk packet, the 'atlk' code makes a LAP Read Dispatch (LRdDispatch) call to the LAP Manager to indicate that a packet needs to be delivered. The 'atlk' code delivers this packet by providing and executing routines that emulate ALAP's ReadRest and ReadPacket routines.

Refer to Chapter 4 for more information on the LAP Manager.

## AppleTalk Address Resolution Protocol (AARP)

AARP can be used to map between any two sets addresses. The AARP implementation that EtherTalk uses maps between a 48-bit Ethernet address and an 8-bit AppleTalk address. To distinguish between these two sets of addresses further, this document will refer to them as follows:

- An Ethernet address, which is the node address that is determined by the Physical and Link layers of the network. An example of an Ethernet address is a 48-bit Ethernet destination address. The Ethernet address is the EtherTalk equivalent of the generic hardware address.
- An AppleTalk address, which is the node address used by high-level AppleTalk protocols. An example of an AppleTalk address is an 8-bit AppleTalk node address for the Datagram Delivery Protocol (DDP). The AppleTalk address is the EtherTalk equivalent of the generic protocol address.

### AARP Functions

A generic AARP implementation resides between the Link Access layer and the Network layer of the network and performs three basic functions:

- *Initial determination of a unique protocol address for a node using a given protocol set.* This address must be unique among all nodes on the network.
- *Mapping from a protocol address to a hardware address.* Given a protocol address for a node on the network, AARP returns either the corresponding hardware address or an error that indicates that no node on the network has such a protocol address.
- *Filtering of packets.* For all data packets received by a given node, AARP verifies that the destination node address of the packet is equal to either the node's protocol address or the network broadcast value or multi-cast value of the node. If the packet does not equal either of these values, AARP discards the packet.

Refer to Chapter 5 for more information on AARP.

## The Ethernet Driver

While using EtherTalk software on the Macintosh II, the Ethernet driver serves as a general-purpose interface between the 'atlk' resource and the Ethernet Interface card. The driver interface is recommended for use with other Ethernet implementations, such as an interface to a Macintosh SE driver.



CONFIDENTIAL

The Ethernet driver, located in the system file, is named .ENET. If you are developing a driver for use with a *slotless* device, name the driver .ENET0.

## Opening the Ethernet Driver

On the Macintosh II, use the Device Manager to make a PBOpen call to open the Ethernet driver. Before you can make this call, you will have to obtain certain field values, such as the EtherTalk card slot number. You may obtain these field values by using the Slot Manager sNextsRsrc trap.

The Ethernet driver opens in AppleTalk mode. In this mode, packets for transmission can contain no more than 768 bytes. Packets for transmission and reception share a common buffer pool. The transmission-packet size is large enough to encapsulate packets for transmission and to allow a larger buffer pool area for packet reception. If packets require more than 768 bytes, issue a control call (ESetGeneral) to change the mode from AppleTalk to General. In General mode, the driver can transmit any valid Ethernet packet.

## Transmission and Reception

A series of Device Manager control calls are made to the driver to control packet transmission and reception over Ethernet. These calls are as follows:

- EAttachPH, which attaches a protocol handler to the driver specified by the protocol type
- EDetachPH, which removes a protocol handler from the driver for the given protocol type
- EWrite, which writes a packet out to Ethernet
- ERead, which reads in a packet
- ERdCancel, which cancels a specified ERead call
- EGetInfo, which returns the node address on which the driver is running
- ESetGeneral, which switches the driver from AppleTalk to General mode

◆ *Note:* For more information about the Ethernet driver, refer to Chapter 6.

# Chapter 3

# Calls to the 'adev' File

This chapter contains information about making calls to the 'adev' file for an AppleTalk selection. The 'adev' file is similar to the 'cdev' file, and both reside in the System Folder. When the user selects the Network 'cdev' icon from the Control Panel, the Network 'cdev' makes a series of calls to each 'adev' file and displays the 'adev' icons to represent all AppleTalk selections available to the user. In addition, the Network 'cdev' file highlights the current AppleTalk selection. If the user then makes a different AppleTalk selection, the Network 'cdev' highlights the new selection and updates parameter RAM with the information obtained from the 'adev' resource. The next time the Macintosh restarts, the LAP Manager INIT resource obtains the latest user selection from parameter RAM, loads the corresponding AppleTalk 'atlk' resource into the system heap, and initializes the 'atlk' code.

## The 'adev' File Contents

The 'adev' file for EtherTalk and any other AppleTalk selection is located in the System Folder and must contain the following resources and code segments as shown in Table 2-1.

| Code | ID |
|---|---|
| 'BNDL' resource | -4032 |
| 'FREF' resource | -4032 |
| 'ICN#' resource | -4032 |
| 'STR ' resource | -4032 |
| 'adev' code segment | in range of 1–254 |
| 'atlk' code segment | in range of 1–254 |

MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 3-1**
'adev' File Contents

In addition to these resources and code segments, the 'adev' file should contain an owner resource to display the icon in the Finder. For example, because the EtherTalk 'adev' file has a creator of etlk, the 'adev' contains an owner resource called *'etlk'* with an ID of 0. In addition, the 'adev' file has its bundle bit set to allow the 'ICN#' resource to display the EtherTalk icon in the Finder.

## The 'adev' and 'atlk' Resources

The 'adev' resource located in an 'adev' file, is responsible for handling all interaction with the user. The Network 'cdev' loads the 'adev' resource into the application heap, calls the 'adev' resource, and removes it as needed. The 'atlk' resource is responsible for the actual implementation of the alternate AppleTalk selection. When the user selects an alternate AppleTalk icon, the Network 'cdev' file loads the 'atlk' resource into the system heap, calls the 'atlk' resource for initialization, and then detaches it. At startup time, the LAP Manager INIT resource performs the loading, calling, and detaching of the 'atlk' resource.

The 'atlk' resource *must* have its system-heap bit set and both the 'adev' and 'atlk' resources should have their locked bit set. The resource ID of the 'adev' resource and the 'atlk' resource *must* be the same and in the range of 1 to 254. When stored in the low byte of parameter RAM, this ID identifies the current AppleTalk selection.

❖ *Note:* Parameter RAM contains 4 bytes of information that identify an AppleTalk selection. The low byte contains the resource ID of the 'adev' resource and the'atlk' resource, and the high bytes contain other information that uniquely identifies the selection.

Like drivers, no two AppleTalk implementations can have the same ID. Apple reserves the use of the ID ranges of 1 to 127. You may use the ID ranges of 128 to 254. Contact Apple Technical Support to obtain an ID.

## Calls to the 'adev' Resource

The Network 'cdev' calls the 'adev' resource, at the first location in the resource (an offset of 0), at two different times:

- When the user selects the Network 'cdev' icon, the Network 'cdev' calls GetADEV.
- When the user selects an alternate AppleTalk icon, the Network 'cdev' calls SelectADEV.

The Network 'cdev' passes a value in register D0 that distinguishes between these two calls. Your code must observe Pascal register saving conventions and should return with an RTS.

## The GetADEV Call (D0 = 101)

| | | |
|---|---|---|
| Call: | D1 (long) = | current value of parameter RAM |
| | D2 (long) = | value returned from previous GetADEV call, or 0 if first GetADEV call |
| Return: | D0 (byte) = | status flag |
| | D2 (long) = | next value for D2 to call; also used by SelectADEV call |
| | | A0 —> string to place under icon. |

When the user selects the Network 'cdev' icon, the Network 'cdev' needs to display a list of icons to represent all alternate AppleTalk selections that are available to the user. To do this, the Network 'cdev' makes a series of GetADEV calls to each 'adev' resource in the System Folder. Since each 'adev' resource could possibly be handling multiple interface cards, the 'adev' resource must tell the Network 'cdev' how many icons to display and identify the string for each icon. The Network 'cdev' displays these identical icons for each card as the 'ICN#' resource specifies. You may use the string to which A0 points to identify the icon uniquely. For example, you could obtain the slot number of the card with a Slot Manager sNextsRsrc call and then append the slot number to the string.

The first GetADEV call contains the current value of parameter RAM in D1, to indicate the current AppleTalk selection, and 0 in D2 to indicate that this call is the first GetADEV call. The 'adev' resource responds to the *first* GetADEV call by returning a status-flag value in D0, indicating whether or not there are additional cards that this 'adev' resource supports. Also, the 'adev' resource returns a value in D2 that the Network 'cdev' associates with this icon, and a pointer in A0 that points to the Pascal string to place under the icon.

❖ *Note:* The Network 'cdev' also passes the D2 value to the 'adev' resource when making the SelectADEV call.

If the status flag indicates to the Network 'cdev' that there is an additional card that the 'adev' resource supports, the Network 'cdev' makes another GetADEV call with the same value in D1 and the D2 value that was returned from the first call. Upon receipt of the D2 value, the 'adev' resource knows it returned first-call information the last time, and responds by returning second-call information to the Network 'cdev'. The Network 'cdev' continues to make subsequent GetADEV calls until the status flag indicates that there are no more cards to support. When the user selects an icon that the 'adev' resource supports, the Network 'cdev' makes a SelectADEV call to the 'adev' resource, passing the value in D2 to indicate the current selection.

### Status-flag Byte

There are three status-flag bytes (–1, 0, and 1) that the 'adev' resource may return in D0 to indicate the status of the alternate AppleTalk selection.

The 'adev' returns D0 = –1 to inform the Network 'cdev' that there is one and maybe more AppleTalk selections (cards) supported by this 'adev' resource. Returning D0 = –1 also indicates to the Network 'cdev' that this AppleTalk selection *seems* to be the one currently selected, as indicated by parameter RAM. The Network 'cdev' responds by making another GetADEV call to the 'adev' resource.

Returning D0 = 0 also informs the Network 'cdev' that there is one and maybe more AppleTalk selections to support; however, returning D0 = 0 also indicates that this AppleTalk selection is *not* the one currently selected, as indicated by parameter RAM. The Network 'cdev' responds by making another GetADEV call to the 'adev' resource.

Returning D0 = 1 informs the Network 'cdev' that there are no more AppleTalk selections to support.

- *Note:* The 'adev' resource may return D0 = 1 in response to the the first GetADEV call to inform the Network 'cdev' that there are currently no AppleTalk selections to support.

Before the 'adev' returns information about an alternate AppleTalk it supports, the 'adev' resource examines the high 3 bytes of the long word in D1 for the current value of parameter RAM. Depending on the contents of D1, the 'adev' resource returns the appropriate status value in D0. If the 'adev' resource returns D0 = –1, the Network 'cdev' checks various system parameters and highlights the icon *only* if it is the current selection. The 'adev' may be wrong about identifying the current selection to the Network 'cdev'. For instance, after two different AppleTalk implementations examine the high 3 bytes of parameter RAM, they both may return D0 = –1. To handle this possibility, the Network 'cdev' examines the low byte of parameter RAM, which contains the resource ID of the previous selection, and matches the ID with the proper AppleTalk implementation. The Network 'cdev' highlights the appropriate icon after making the final determination.

### The SelectADEV Call (D0 = 102)

Call: D2 (long) = value returned from associated GetADEV call.

Return: D1 (high three bytes) = value to set in parameter RAM; also passed to 'atlk' code by the AInstall call.

The Network 'cdev' makes a SelectADEV call to the associated 'adev' resource when the user selects an alternate AppleTalk icon. This call's main purpose is to determine the value that the 'atlk' code wishes to store in parameter RAM. This value, which is specific to the alternate AppleTalk implementation, indicates the details of the alternate AppleTalk selection and is passed to the 'atlk' resource by the AInstall call. For instance, in addition to the resource ID of the 'adev' resource and 'atlk' resource, the high 3 bytes may contain the slot number of the interface card ($09 – $0E). Depending on your application, you may wish to direct the 'adev' resource to display a dialog box at this point to obtain some type of user information such as data rate, and to save this information in parameter RAM.

◆ *Note:* The SelectADEV call is not an initialization call.

## Calls to the 'atlk' Resource

The 'atlk' resource, loaded into the system heap by the Network 'cdev', contains two distinct sections of code. The first section, at the start of the resource, contains the LAPWrite code to be inserted into the LAPWrite hook as the alternate AppleTalk implementation. This procedure is explained in detail in Chapter 4.

The second section of the code, located at the start of the 'atlk' resource plus two, contains the initialization and shutdown routines. After the Network 'cdev' makes the SelectADEV call to the 'adev' resource, the Network 'cdev' loads the associated 'atlk' resource into the system heap and calls it with AInstall to perform initialization. The LAP Manager INIT resource also makes the AInstall call at startup time to initialize the 'atlk' resource as indicated by parameter RAM. Following this call, if there is no error, the Network 'cdev' detaches the 'atlk' resource (from the Resource Manager) so it will remain in the system heap when the user closes the Control Panel.

The Network 'cdev' also makes an AShutdown call to dispose of the previously selected 'atlk' resource. Before making this call, the Network 'cdev' needs to obtain the location of the 'atlk' resource because it is detached from the Resource Manager. To accomplish this, the Network 'cdev' calls the LAP Manager with LWrtGet, which returns the location of the LAPWrite code. The LAPWrite code starts at the beginning of the 'atlk' resource; therefore, the Network 'cdev' knows where to call the 'atlk' code with AShutdown.

For the AInstall and AShutdown calls, the contents of register D0 indicate which call is made by the Network 'cdev'. The code must observe interrupt-register-saving conventions (it may use D0–D3 and A0–A3) and should return with an RTS.

## The AInstall Call (D0 = 1)

Call: D1 (long) = value from parameter RAM (as set in the SelectADEV call)

Return: D0 = error code.

D1 (high 3 bytes) = new value to set in parameter RAM

When the Network 'cdev' or LAP Manager INIT resource makes the AInstall call to the 'atlk' code, it should respond by allocating variables, opening the appropriate I/O device (such as the slot driver), and performing any other initialization necessary. The 'atlk' code should call the LAP Manager with a LWrtInsert call to install itself as the alternate AppleTalk selection. This call should return a value to set in parameter RAM only if that value is different than the one received; otherwise, the 'atlk' code should preserve D1. If an error occurs during any portion of this process, your code should return a negative value in D0; otherwise, D0 should return 0.

## The AShutdown Call (D0 = 2)

There are no arguments to the AShutdown call. The Network 'cdev' makes this call after the LAP Manager closes the AppleTalk drivers, and before the Network 'cdev' installs a new alternate AppleTalk implementation. The 'atlk' code should issue a LWrtRemove call to the LAP Manager, dispose of its variables, and perform any other operations necessary before the Network 'cdev' disposes of the 'atlk' resource.

# Chapter 4

# Calls to the LAP Manager

The LAP Manager standardizes interactions between the AppleTalk protocol stack and the Link Access layer of the current AppleTalk selection. Standardizing these interactions ensures that various AppleTalk implementations will not interfere with each other and will not have to make use of information that is private to the AppleTalk drivers. The LAP Manager resides between the LAPs (such as ALAP and ELAP) of all AppleTalk implementations and the AppleTalk protocol stack.

This chapter describes the calls that the LAP Manager provides. Once the Network 'cdev' loads the 'atlk' code into the system heap and makes the AInstall call, the 'atlk' code responds by making a call to the LAP Manager which inserts the 'atlk' code into the LAPWrite hook. The LAP Manager also provides functions for removing the 'atlk' code from the LAPWrite hook, receiving packets from the network, and standardizing the packet transfer process with AppleTalk's .MPP driver.

The LAP Manager is installed in the system heap at boot time, before the AppleTalk Manager opens the .MPP driver.

## Calling the LAP Manager

The 'atlk' code resource makes all calls to the LAP Manager by jumping through a low-memory location, with D0 equal to a dispatch code that identifies the function. The exact sequence is

```
MOVE.W    #Code,D0          ; D0 = function
MOVE.L    LAPMgrPtr, An     ; An -> start
JSR       LAPMgrCall (An)   ; Call at entry point
```

LAPMgrPtr is defined as the low-memory global ATalkHk2, which is the location jumped through by the .MPP driver immediately before it writes a packet out through ALAP to the APN. If the user selects an alternate AppleTalk implementation, the LAP Manager uses LAPMgrPtr to take control at this point and call the alternate AppleTalk implementation. Offset LAPMgrCall within this code is the command-processing part of the LAP Manager.

- *Note:* ATalkHk2 is not defined in the original Macintosh ROMs. The LAP Manager is available only on Macintosh Plus and later ROMs.

## LAP Manager Functions

The LAP Manager supports the following nine functions that are used for packet handling.

### LWrtInsert (D0 = 2)

Call: A0 —> code to insert (first part of 'atlk' resource)
D1 (byte) = flags
D2 (word) = maximum number of times to try to get an unused node address (0 = infinite)

Return: D0 = 0 (no error)

This call inserts an alternate AppleTalk in the LAPWrite hook. After the 'atlk' resource makes this call, the LAP Manager calls the code to which A0 points before writing any packet out on the network. Use the bits in the low byte of D1 to inform the LAP Manager of the way to handle the packet. Set these bits to indicate the following to the LAP Manager:

- *Bit 7* = let the 'atlk' code handle self-sends (intranode delivery); normally the LAP Manager intercepts self-send packets and processes them.
- *Bit 6* = do not disable the port B serial-communications controller (SCC); normally the LAP Manager disables the SCC.
- *Bit 5* = honor the server/workstation (server/wks) bit in the node-number-assignment algorithm.

The LAP Manager generally handles intranode-packet delivery (packets sent to one's own node). If a packet is an intranode packet, the LAP Manager delivers this packet without calling the code in the LAPWrite hook; however, if the packet is a broadcast, the LAP Manager delivers the packet within its node and calls the 'atlk' code to handle the broadcast delivery. For this process to happen, the .MPP driver's SelfSend flag must be set. To disable the LAP Manager's handling of intranode delivery, set bit 7 in D1 when making the LWrtInsert call.

Setting bit 6 in D1 tells the LAP Manager *not* to disable SCC port interrupts. Normally, the LAP Manager disables these interrupts because it assumes that the alternate AppleTalk implementation does not want to receive ALAP packets on this port.

When picking a node address, set bit 5 to tell the LAP Manager to honor the server/wks bit. Normally, the LAP Manager assumes that the alternate AppleTalk implementation does not distinguish between server addresses (128–254) and workstation addresses (1–127), and that the AppleTalk implementation wants to pick a node address in the full range of 1 to 254.

The LAP Manager calls the code to which A0 points at two different times. The first is at node-address-choosing time. The LAP Manager calls the 'atlk' code for each set of ENQs (ALAP type $81) that ALAP would normally send out to the network. The second is at the time when the AppleTalk drivers would normally write a packet out through ALAP. Once installed in the LAPWrite hook, the LAP Manager calls the 'atlk' code as follows:

A0 —> where to return when done with the operation

A1 —> WDS (if sending a data packet, not an ENQ) or port-use byte (if sending ENQs)

A2 —> .MPP variables

D0 (byte) = nonzero if sending ENQs, zero if not

D1 —> where to return in .MPP to continue packet processing

D2 (byte) = ALAP destination address

The 'atlk' code should return with a normal RTS if the write is still in progress, and should jump to the location to which A0 points when the write finishes. When the write finishes, it must reset A1, A2, and D2 to their initial values, and must preserve A4–A6 and D4–D7. If the code wishes the .MPP driver to continue its normal processing (for example, if the code does *not* intercept the call), it should jump to the location to which D1 points. Generally, code will intercept the call.

If D0 is nonzero, which indicates a call to send ENQs, the code should query if the address that D2 specifies is in use, and return through A0 immediately. At any time thereafter, if the code discovers that the address is in use, the code should make a LSetInUse call to the LAP Manager.

❖ *Note:* The LAP Manager passes both the variable pointer of the .MPP driver and the address of the port-use byte (if sending ENQs) to the 'atlk' code. Do *not* assume that pointer is stored at location $2D8 (AbusVars) or that the port-use byte is at location $291 (PortBUse). Save these pointers from the first ENQ call for future use.

## LWrtRemove (D0 = 3)

Return: D0 = 0 (no error)

The 'atlk' code makes the LWrtRemove call to the LAP Manager to remove an alternate AppleTalk selection from the LAPWrite hook. Generally, the 'atlk' code should make this call following an AShutdown call.

## LWrtGet (D0 = 4)

Return: D0 = 0 (no error)
A0 —> start of code in the LAPWrite hook

When the LWrtGet call is made, A0 returns a pointer to the alternate AppleTalk code in the LAPWrite hook. Normally, the 'atlk' code does not need to make this call; however, the Network 'cdev' makes this call as part of the process to dispose of the 'atlk' code.

## LSetInUse (D0 = 5)

Call: A2 —> .MPP variables
Return: D0 = 0 (no error)

This LSetInUse call indicates to the LAP Manager and the .MPP driver that another node on the network is currently using the requested node address. The .MPP driver will try another address.

## LGetSelfSend (D0 = 6)

Call: A2 —> .MPP variables
Return: D0 = 0 (no error)
D1 (byte) = value of .MPP SelfSend flag

This LGetSelfSend call is for use by alternate AppleTalk implementations that use their own intranode delivery. If D1 is nonzero, intranode delivery is enabled.

## LRdDispatch (D0 = 1)

Call: A2 —> .MPP variables

Return: D0 = non-zero if error

This LRdDispatch call indicates to the LAP Manager that a packet has arrived from the network and requires delivery. Registers should be set up to provide a simulation of the ALAP client ReadPacket and ReadRest routines. Refer to Chapter 10 *Inside Macintosh*, Volume II for details. Specifically, register setup and restrictions are as follows:

A0, A1 —> hardware register (can be used by the alternate AppleTalk for any reason)

A2 —> .MPP variables

A3 —> past the 5 header bytes in the .MPP RHA

A4 —> the ReadPacket routine (previous value saved and restored after ReadRest is complete)

A5 has been saved and is restored after ReadRest is complete

D1 = packet length left to input

D2 (byte) = LAP type for which to dispatch a protocol handler

◆ *Note:* The ReadRest routine begins 2 bytes after ReadPacket.

Generally the LRdDispatch routine, even though it is called with a JSR, will not return to the caller, but will jump to the protocol handler attached to the protocol indicated in D2, which in turn calls ReadPacket and ReadRest routines. If the routine does return, doing so indicates an error—there was no handler attached to the protocol indicated in D2.

## LGetATalkInfo (D0 = 9)

Return: D1 (long) = value of parameter RAM

Uses A0

This LGetATalkInfo call returns the current 4-byte value of parameter RAM. The low byte contains the resource ID of 'adev' resource and the 'atlk' resource for the current AppleTalk implementation (0 for Built-in and 2 for Ethertalk). The high 3 bytes contain values that further distinguish this AppleTalk implementation.

## LAARPAttach (D0 = 7)

Call: D1 (long) = hardware/protocol type (hardware type in high word).
D2 (word) = Ethernet driver reference number
A0 —> listener code

Return: D0 = non-zero if error
Uses A0 and D2

This call is only used when attaching an AARP listener to the LAP Manager to handle incoming AARP packets other than those used to map between Ethernet and AppleTalk addresses. The LAP Manager determines which AARP listener to attach by examining the contents of D1. Currently, the LAARPAttach call only supports one driver.

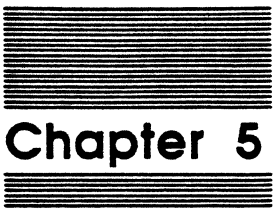
◆ *Note:* The LAARPAttach and LAARPDetach calls are used to multiplex incoming AARP packets for various possible hardware-protocol mappings. These two calls should be used by any application that wishes to receive AARP packets.

## LAARPDetach (D0 = 8)

Call: D1 (long) = hardware/protocol type (hardware type in high word)
D2 (word) = Ethernet driver reference number

Return: D0 = nonzero if error
Uses D2

The LAARPDetach call detaches an AARP listener as the contents of D1 specify.



# Chapter 5

# AARP and Data Packets

Depending on your application, you may decide to use AARP to resolve your network addressing requirements. This chapter details the operation of AARP from a generic standpoint and also explains the way EtherTalk uses AARP to resolve network-addressing requirements. In addition, this chapter discusses generic AARP packet formats, EtherTalk AARP packet formats, and the EtherTalk data packet format.

# About AARP

Basically, AARP is a set of rules and procedures that work together to provide packet-addressing information to an AARP client. To ensure proper and efficient packet delivery and reception on the network, AARP maintains a collection of protocol addresses and their corresponding hardware addresses for each protocol set that a node supports.

## Protocol Sets

A protocol set is a collection of related protocols that correspond to the layers of the ISO-OSI reference model. Protocol sets enable transmission and reception of packets over a network. AppleTalk protocols are an example of a protocol set. Information on the network is transferred between protocol sets of the same type. For example, when a node transmits an AppleTalk packet, the node addresses the packet to a receiving node's AppleTalk protocol set. Before a node sends a packet on the network, the sending node addresses the packet to the recipient by inserting a hardware destination address and a protocol address into the header section of the packet. These two addresses, when used together, identify the node that is to receive the packet and the protocol set for which the packet is indended. Because a node may support more than one protocol set, AARP maintains a collection of protocol-to-hardware address mappings for each protocol set that a node supports. These address mappings are kept in an address mapping table (AMT) which is updated by AARP to ensure that current addressing information is available. The AMT serves as a cache of known protocol-to-hardware address mappings. The way AARP obtains these protocol-to-hardware address mappings is explained later in this chapter.

## Hardware Addresses

A hardware address is the address that is determined by the Physical and Data Link layers of the network. Each node on the network must have a hardware address that is unique. An example of a hardware address is a 48-bit Ethernet node address or an 8-bit AppleTalk Link Access Protocol address. In addition to receivng packets addressed to a node's hardware address, a node may also receive packets that are addressed to the node's *broadcast-hardware* address or a *multicast* address. If a sending node transmits a packet that contains a hardware-broadcast address as the destination address, all nodes on the network will receive the packet. The hardware-broadcast address is predefined by the network and is the same for all nodes on the network. A multicast address is similar to a broadcast-hardware address. If a sending node transmits a packet that contains a multicast address as the destination address, only a specific subset of all nodes on the network will receive the packet. Depending on network configuration and application, some nodes on the network may *not* have a multicast address, and other nodes may have one or more multicast addresses. In summary, each node on the network will receive all packets sent to the node's unique hardware address, to the broadcast-hardware address, and to any multicast address group to which the node belongs.

## Protocol Addresses

A protocol address is the address that a node assigns to identify the protocol client that is to receive a packet for a given protocol set. An example of a protocol address is the 8-bit AppleTalk protocol address that the Datagram Delivery Protocol (DDP) and AARP use to verify that an incoming packet is intended for this DDP. For EtherTalk, AARP randomly assigns a protocol address at initialization time and verifies that this protocol address is unique among all other protocol addresses on the network. Once AARP verifies that this address is unique, AARP informs DDP of the protocol address. In addition to receiving packets that contain a unique protocol protocol address, a protocol client (such as DDP) may also receive packets addressed to a *broadcast-protocol* address. As the broadcast-hardware address causes all nodes on the network to respond at the physical level, the broadcast-protocol address causes all nodes on the network to respond at the "protocol-set" level. For example, addressing a packet with a broadcast-hardware address and a broadcast-protocol address for the AppleTalk protocol set causes all nodes on the network to receive the packet. However, only those nodes that support the AppleTalk protocol set will process the packet. If a node supports more than one protocol set, this node (or AARP) should assign a protocol address that corresponds to a protocol client for each protocol set.

## Obtaining an Address

Generally to send packets on the network, a transmitting AARP client requests from AARP the hardware address that corresponds to the protocol address of the node that is to receive the packet. To provide its client with the desired hardware address, AARP attempts to retrieve, from the cache of address in the AMT, this hardware address. If the hardware address is among the cache, AARP returns this address to its client. If the protocol-to-hardware mapping is not resident, AARP transmits a series of AARP packets to all nodes on the network to obtain the desired hardware address.

# AARP Functions

A generic AARP implementation resides between the Link Access layer and the Network layer of the network and performs three basic functions for each protocol set that AARP supports:

- *Initial determination of a unique protocol address for a given protocol client.* This address must be unique among all nodes on the network.
- *Mapping from a protocol address to a hardware address.* Given a protocol address for a node on the network, AARP returns either the corresponding hardware address or an error that indicates that no node on the network has such a protocol address.
- *Filtering of packets.* For all data packets received by a given node, AARP verifies that the destination protocol address of the packet is equal to either the node's protocol address or the broadcast-protocol value. If the packet does not equal either of these values, AARP discards the packet.

# Packet Categories

Within a given protocol set, there are two categories of packets that a node may encounter on a network. This document distinguishes one category as AARP packets and the other category as data packets. AARP packets are those packets that perform address-resolution functions (such as request, response, and probe). Data packets are those packets that contain information for processing by a protocol set.

## EtherTalk Addresses

AARP can be used to map between any two sets addresses. The AARP implementation that EtherTalk uses maps between a 48-bit Ethernet address and an 8-bit AppleTalk address. To distinguish between these two sets of addresses further, this document will refer to them as follows:

- An Ethernet address, which is the node address that is determined by Ethernet's Physical and Data Link layers of the network. An example of an Ethernet address is a 48-bit Ethernet destination address. This document uses the term "Ethernet address" to refer to the EtherTalk implementation of a generic hardware address.
- An AppleTalk address, which is the node address used by high-level AppleTalk protocols. An example of an AppleTalk address is an 8-bit AppleTalk address that AARP and DDP use to ensure that a packet is intended for processing by this DDP. This document uses the term "AppleTalk address" to refer to the EtherTalk implementation of a generic protocol address.

## AARP Operation

The following section details the operational concept of AARP. These sections contain generic AARP information followed by an EtherTalk example.

### The Address Mapping Table

Within a given node, AARP maintains an Address Mapping Table (AMT) for each protocol set that a node supports. Each AMT contains a list of protocol addresses and their corresponding hardware addresses—serving as a cache of known protocol-to-hardware address mappings. Whenever AARP learns of a new mapping, AARP updates the appropriate AMT to reflect the new addresses. If there is no more room for new addresses in an AMT, AARP should purge this AMT by using some sort of least-recently used algorithm.

- *EtherTalk example:* Within a given node, AARP maintains an Address Mapping Table (AMT) for the AppleTalk protocol set. The AMT for EtherTalk contains a list of AppleTalk addresses and their corresponding Ethernet addresses—serving as a cache of known AppleTalk-to-Ethernet address mappings. Whenever AARP learns of a new mapping, AARP updates the AMT to reflect the new addresses. Note that AppleTalk addresses are 8 bits in length, thefore, the AMT will contain no more than 256 address entries.

## Choosing an Address

Each protocol set supported by a node must have an associated protocol address. This address is usually assigned at initialization time. AARP includes one way of making this assignment; however, your AARP client may choose to assign its own protocol address and inform AARP of this address using a different method. The only requirement is that these protocol addresses are unique for each protocol set.

- *EtherTalk example:* At initialization time, AARP randomly picks an AppleTalk address for the node that AARP supports. After checking with other nodes on the network to ensure that this address is unique, AARP assigns this address as the node's AppleTalk address.

## Random Address Selection

AARP includes the ability to pick a unique protocol address dynamically at initialization time. When an AARP client requests this function, AARP picks a protocol address at random for a given protocol set, and sets that address as the node's tentative protocol address. If, by chance, there is already a mapping for that address in the AMT for that protocol set, AARP knows that another node on the network is using this protocol address. AARP continues to pick additional random addresses until it identifies an address that is not in the AMT. Once AARP identifies an address that is not in the AMT, AARP verifies the uniqueness of the address as described in the following chapters.

- *EtherTalk example:* When an AARP client requests an AppleTalk address, AARP picks an AppleTalk address at random and sets that address as the node's tentative AppleTalk address. If, by chance, there is already a mapping for that address in the AMT, AARP picks additional random addresses until it identifies an address that is not in the AMT.

## Probe Packets

Once AARP identifies a tentative protocol address for a given protocol set, AARP broadcasts a number of *probe packets* that contain the tentative protocol address (for a given protocol set) to determine if any other node on the network is currently using that protocol address. Any node receiving a probe packet whose protocol address matches its protocol address must respond by sending an AARP response packet.

- *EtherTalk example:* Once AARP identifies a tentative AppleTalk address, AARP broadcasts a number of probe packets that contain the tentative AppleTalk address to determine if any other node on the network is currently using that AppleTalk address. Any node receiving a probe packet whose AppleTalk address matches its AppleTalk address must respond by sending an AARP response packet.

## Response to Probe Packets

When a node receives a probe packet for a protocol set that this node supports, it checks its protocol address that is associated with the protocol set . If the tentative protocol address matches the the receiving node's protocol address, the receiving node sends an AARP response packet to the probing node. Upon receiving the response packet, the probing node knows the protocol address is already in use and probes with another address. If the probing node does *not* receive a response packet after a specific number of probes, AARP sets the tentative protocol address to permanent and returns this address to its client.

- *EtherTalk example:* When a node receives an AARP probe packet, this node matches this address to its AppleTalk address. If the AppleTalk addresses match, the receiving node sends an AARP response packet to the probing node. Upon receiving the response packet, the probing node knows the AppleTalk address is already in use and probes with another address. If the probing node does *not* receive a response packet after a specific number of probes, AARP sets the tentative AppleTalk address to permanent and returns this address to its client.

## Avoiding Duplicate Tentative Addresses

It is possible, although unlikely, that two nodes on the network could pick the same tentative address at the same time. To avoid this possibility, if a node receives a probe packet whose tentative address matches its tentative address, the receiving node should assume that this address is in use and select another random address. A node should never respond to an AARP probe packet or an AARP request packet while it is probing.

## Request Packets

When an AARP client makes a request to determine the hardware address that corresponds to a protocol address for a given protocol set, AARP first scans the associated AMT for the protocol address. If the protocol address is in the AMT, AARP returns the corresponding hardware address. If the hardware address is *not* in the AMT, AARP attempts to determine the hardware address by broadcasting a series of AARP *request packets* to all nodes on the network. The request packet indicates the protocol address for which a hardware mapping is desired, as well as the type of protocol set for that mapping.

- *EtherTalk example:* When the AARP client makes a request to determine the Ethernet address that is associated with an AppleTalk address, AARP first scans the AMT for the AppleTalk address. If the AppleTalk address is in the AMT, AARP returns the corresponding Ethernet address. If the Ethernet address is *not* in the AMT, AARP attempts to determine the Ethernet address by broadcasting a series of AARP request packets to all nodes on the network. The request packet indicates the AppleTalk address for which an Ethernet mapping is desired.

## Response to Request Packets

When a node receives a request packet, AARP attempts to match the desired protocol address to its own protocol addresses for the given protocol set. If the receiving node's protocol address for that protocol set matches, the receiving node responds by sending an AARP response packet to the requestor indicating the protocol-to-hardware node-address-mapping information. The requesting AARP enters this mapping in the AMT and returns the hardware address to AARP's client. If there is no reply within a specific time-out period, AARP retransmits the packet a given number of times and returns an error to its client if there is still no response; the error indicates there is no such node on the network.

- *EtherTalk example:* When a node receives an AARP request packet, the node's AARP attempts to match the desired AppleTalk address to its own AppleTalk address. If the receiving node's AppleTalk address matches, the receiving node responds by sending an AARP response packet to the requestor. The response packet contains the receivng node's Ethernet address. The requesting AARP enters this mapping in the AMT and returns the hardware address to AARP's client. If there is no reply within a specific time-out period, AARP retransmits the packet a given number of times and returns an error to its client if there is still no response; the error indicates there is no such AppleTalk node on the network.

## Examining Incoming Packets

In addition to receiving and processing its own packets (probe, request and response), an active AARP (such as one that is performing translation) should receive and process all packets for each protocol set that AARP supports. There are two reasons for this requirement. The first reason is that AARP must verify that an incoming packet is in fact addressed to its client for the given protocol set. The second reason is that AARP can gather or *glean* address information from the incoming packet to update the AMT, limiting the number of AARP packets sent on the network.

- *EtherTalk example:* In addition to receiving and processing its own packets (probe, request and response), AARP receives and processes all AppleTalk data packets to glean packet-address information from the packet and update the AMT. Also, AARP verifies that the incoming packet is intended for the node's AppleTalk address (or broadcast-protocol address) and , if so, passes the packet to the AppleTalk protocol stack for further processing.

### Verifying Packet Address

To verify that an incoming data packet is intended for a client that AARP serves, AARP examines the packet's destination-protocol address. Because the protocol set to which the packet belongs determines the data packets's construction, the location of the destination address within a data packet is different for different protocol sets. AARP's client must inform AARP of the location of the data packet's destination address for each protocol set that AARP supports. The AARP client must also inform AARP of which address or addresses to accept as broadcast-protocol values. If AARP determines that the destination-protocol address of the packet does not match the node's protocol address or broadcast-protocol address, AARP must discard the packet and assume the originator sent this packet by mistake.

- *EtherTalk example:* To verify that an incoming data packet is intended for the AppleTalk address that AARP serves, AARP verifies that the packet's destination address in the ALAP header matches the node's AppleTalk address, broadcast-protocol value ($FF). Figure 5-2 shows the location of the destination-protocol address in the data packet's header. If AARP determines that the destination address of the packet does not match the node's AppleTalk address or broadcast-protocol address, AARP discards the packet.

## Gleaning Information

Incoming data packets will generally contain the source hardware address and the source protocol address. Once AARP determines that the packet contains a valid protocol address, AARP can glean the source hardware and protocol address-mapping information from the packet and update the appropriate AMT. Gleaning mapping information in this fashion eliminates the need to send an additional request packet when the node next tries to communicate with the sender.

Note that this gleaning of source information from client packets is *not* a requirement of AARP. In certain cases, this information may not be available. Depending on your application, you may determine that gleaning information is too inefficient to add an entry to the AMT for each incoming packet.

Source information can also be gleaned from AARP request packets. Because these packets are broadcast to every node on the network, every AARP implementation receives them. These packets always contain the source hardware address and source protocol address. AARP should always add this address information to its AMT, even if AARP does not answer this packet. AARP should *not* glean any source information from probe packets because this information is tentative.

- *EtherTalk example:* Incoming data packets contain the source Ethernet address and the source AppleTalk address. Once AARP determines that the packet is intended for AARP's AppleTalk client, AARP gleans the AppleTalk-to-Ethernet address-mapping information and updates the AMT. Gleaning mapping information in this fashion eliminates the need to send an additional AARP request packet when the node next tries to communicate with the sender.

  Source information is also gleaned from AARP request packets. These packets always contain the source Ethernet address and source AppleTalk address. AARP adds this address information to its AMT, even if AARP does not answer this packet. AARP does not glean source information from probe packets because this information is tentative.

# Aging AMT Entries

An AARP implementation may wish to age AMT entries. One method of doing this is for AARP to associate a timer with each AMT entry. Each time AARP receives a packet that causes an entry update or confirmation in the AMT, AARP resets that entry's timer. If AARP does not reset the entry's timer within a certain period of time, the timer times out and AARP removes this entry from the AMT. The next request for the protocol address associated with this entry will result in AARP sending a request packet, unless AARP gleans a new mapping for this entry after removing it.

Aging AMT entries prevents the following situation: when one node goes down or takes itself off the network, a second node with a different hardware address starts up and acquires the same protocol address as .the first node. An AARP implementation in a third node needs to learn about this change in mapping. Unless the second node broadcasts an AARP request, the third node will not be aware of this change and will continue to contain an invalid hardware address in the AMT.

◆*EtherTalk example:* AARP associates a timer with each AMT entry. Each time AARP receives a packet that causes an entry update or confirmation in the AMT, AARP resets that entry's timer. If AARP does not reset the entry's timer within a certain period of time, the timer times out and AARP removes this entry from the AMT. The next request for the AppleTalk address associated with this entry will result in AARP sending a request packet, unless AARP gleans a new mapping for this entry after removing it.

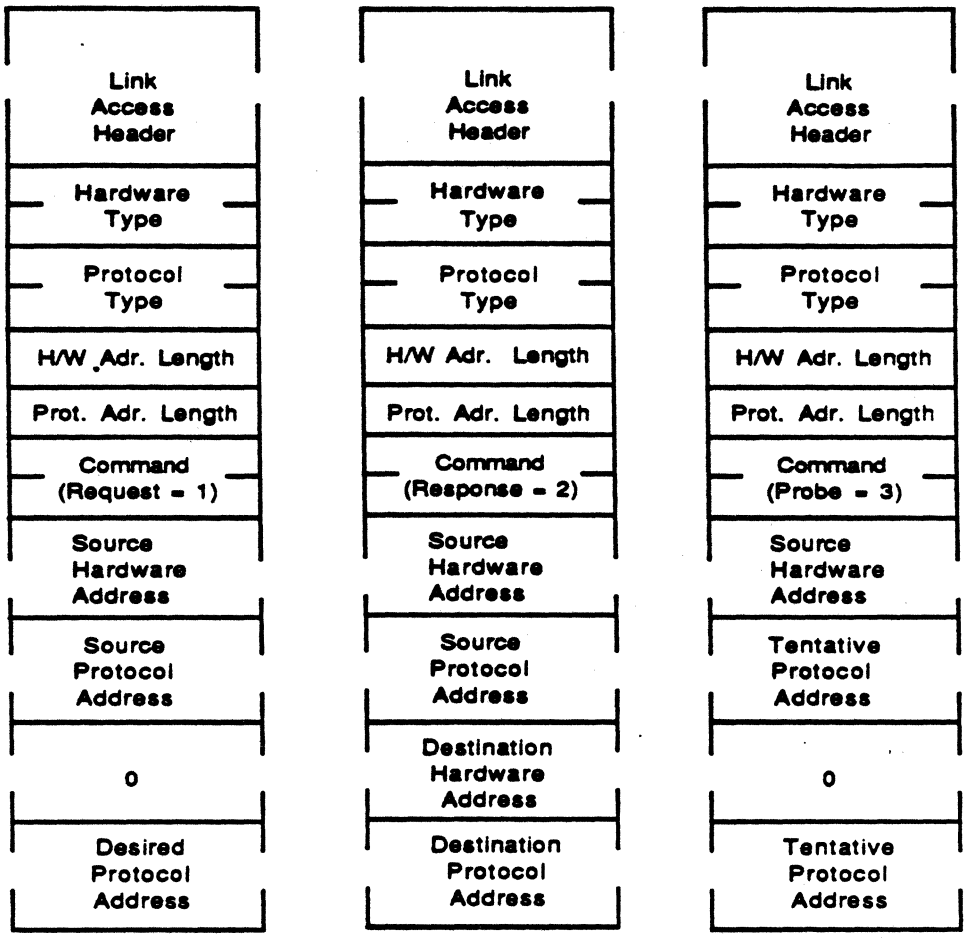
## Age-on-probe

Instead of using timed aging, another approach is to remove an AMT entry whenever AARP receives a probe packet for the entry's protocol address. This process guarantees that the AMT always contains current mapping information, although unnecessary entry removal occurs if a new node probes for an address that is already in use. AARP should implement this age-on-probe function in any node that does not glean address information from data packets because time-based aging in this case is inefficient (data packets would not reset the aging timer).

◆*EtherTalk example:* In addition to timed aging, AARP also incorporates an age-on-probe function. AARP removes an AMT entry whenever AARP receives a probe packet for the entry's AppleTalk address. This process guarantees that the AMT always contains current mapping information, although unnecessary entry removal occurs if a new node probes for an AppleTalk address that is already in use.

## Generic AARP Packet Formats

Refer to Figure 5-1 for the generic AARP packet formats.




AARP Request Packet AARP Response Packet AARP Probe Packet

MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 5-1**
Generic AARP Packet Formats

Each packet begins with the standard link access header for the medium in use (14 bytes for Ethernet). Following this, there is header information which is a constant for the particular protocol-to-hardware mapping. This header information consists of the following:

- Two-byte hardware type, which indicates the medium type (pre-defined).
- Two-byte protocol type, which indicates the desired protocol set (pre-defined).
- One-byte hardware address length, which indicates the length, in bytes, of this field.
- One-byte protocol address length, which indicates the length, in bytes, of this field.

Following this header is a two-byte command field that indicates the packet function (request, response, or probe). Next are the hardware and protocol addresses of the sending node (their length are specified in the preceeding length fields). Last in the packet are the hardware and protocol addresses of the node that is to receive this packet.

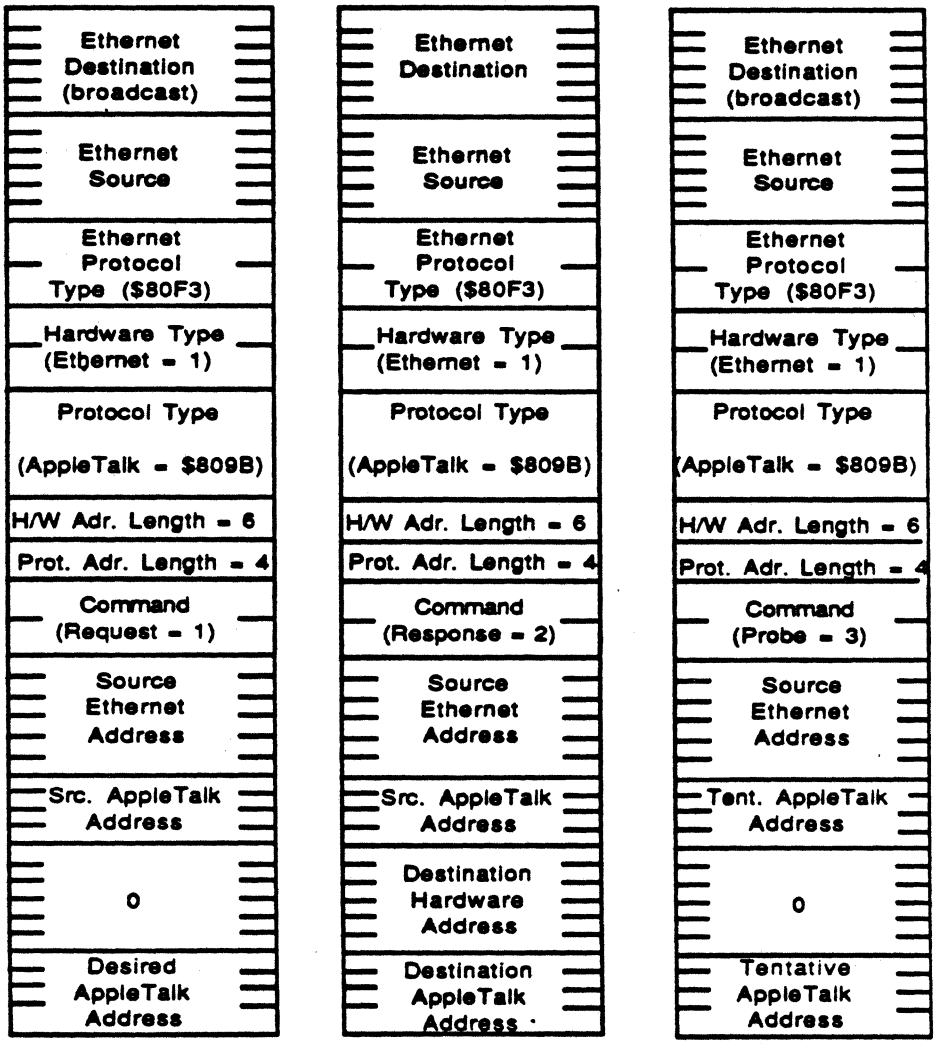
In the case of an AARP request packet, the hardware address of the destination is unknown and should be set to zero. The protocol address should be the address for which a hardware mapping is desired.

For the probe packet, both the source and destination protocol addresses should be set to the sender's tentative protocol address and the destination hardware address should again be set to zero.

❖ *Note:* These conventions also apply to AARP Ethernet-AppleTalk packets.

## AARP Ethernet-AppleTalk Packet Formats

Figure 5-2 shows the AARP Ethernet-AppleTalk packet formats.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 5-2**
AARP Ethernet-AppleTalk Packet Formats

Each AARP packet on Ethernet begins with the Ethernet 14-byte link access header. Following the Ethernet header, there are 6 bytes (predefined) of additional header information that further identify this AARP packet:

- Two-byte hardware type, which indicates Ethernet as the medium
- Two-byte protocol type, which indicates the AppleTalk protocol
- One-byte hardware address length, which indicates the length in bytes of the Ethernet address
- One-byte protocol length, which indicates the length in bytes of the AppleTalk address

Following this header information is a 2-byte command field that indicates the packet function (request, response, or probe). Next are the Ethernet and AppleTalk addresses of the sending node. Last in the packet are the Ethernet and AppleTalk addresses of the node that is to receive this packet.

In the case of an AARP request packet, the Ethernet address of the destination is unknown and should be set to 0. The AppleTalk address should be the address for which an Ethernet address mapping is desired.

For the probe packet, both the source AppleTalk address and destination AppleTalk address should be set to the sender's tentative AppleTalk address and the destination hardware address should again be set to 0.

## Retransmission Details

AARP must retransmit both probes and requests until AARP either receives a reply or exceeds a maximum number of retries. The specifics of the retransmit count and interval depend on the desired thoroughness of the search. In general, the probe-retransmission interval is fixed by AARP, but the request-packet-transmission interval can be assigned as a client-dependent parameter.

## Packet Specifics

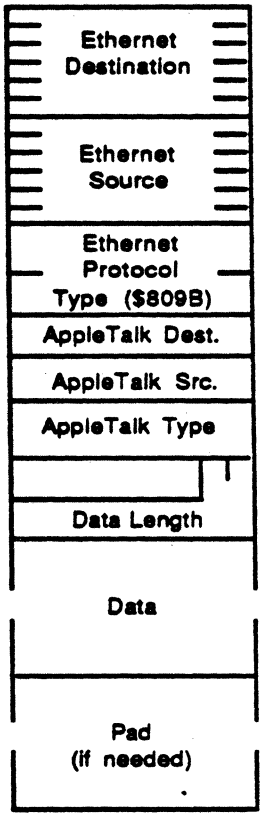
The following constants are currently defined for AARP.

- Protocol type for Ethernet-like media (in data link header): $80F3
- AARP hardware type for Ethernet: $0001
- AARP AppleTalk protocol type: $809B
- AARP Ethernet address length: 6
- AARP AppleTalk address length: 4—first 3 bytes of the address must be 0 and are reserved by Apple for future use
- AARP request command: $0001

- AARP response command: $0002
- AARP probe command: $0003
- AARP probe-retransmission interval for Ethernet-AppleTalk packets: 1/30 second
- AARP probe-retransmission count for Ethernet-AppleTalk packets: 20

## EtherTalk Data Packet Format

Figure 5-3 shows the data-packet format for AppleTalk packets on Ethernet.



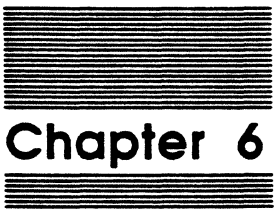

MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 5-3**
EtherTalk Data Packet Format

AppleTalk Packets on Ethernet contain the standard 14-byte header to identify the Ethernet destination, Ethernet source, and Ethernet protocol type. For AppleTalk packets, the Ethernet protocol type is $809B. A complete AppleTalk packet follows this header. The AppleTalk packet consists of a 3-byte header to specify the AppleTalk destination, source, and type, followed by the data field. The low-order 10 bits of the first 2 bytes in the data field contain the length in bytes of the data field (self-including). The high-order 6 bits are protocol dependent.

The minimum size of Ethernet packets is 60 bytes. Including the header, an Ethernet-AppleTalk packet could be as small as 19 bytes; therefore, the packet must be padded to increase packet size to 60 bytes. The contents of the pad are undefined. The maximum size of an AppleTalk packet on Ethernet is 603 bytes plus 14 bytes Ethernet header, or a total of 617 bytes.

Apple recommends, although currently does not require, that any DDP packet sent on Ethernet use the extended DDP header format (see *Inside AppleTalk* for details). This header format ensures compatibility with potential future systems that may require such a header. All EtherTalk implementations must accept extended headers for any incoming AppleTalk packet and these implementations should also accept short DDP headers (including packets which *Inside AppleTalk*, July 1986 identifies as requiring short headers ).



# Chapter 6

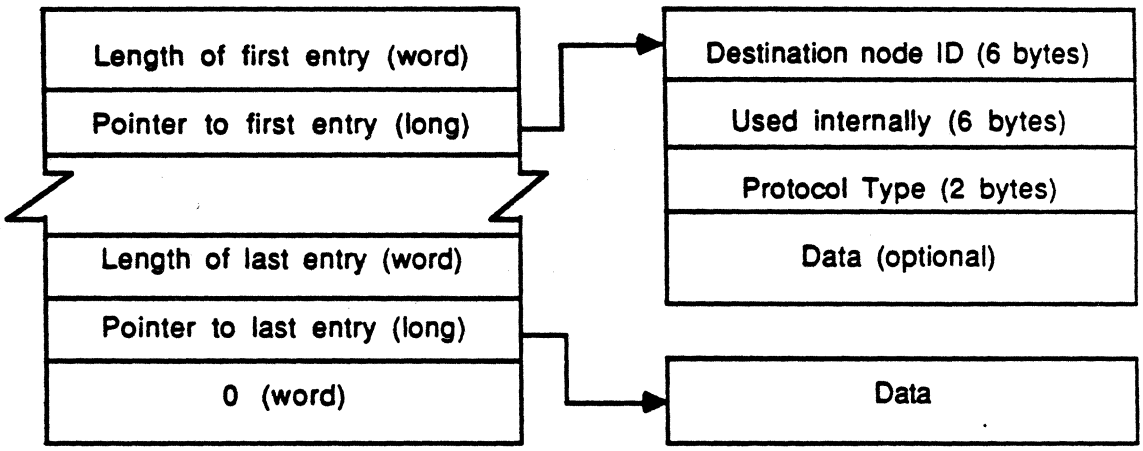
# The Ethernet Driver

EtherTalk software uses a general-purpose Ethernet driver to transmit and receive packets on the Ethernet network. Provided with EtherTalk software, the Ethernet driver is specifically designed for use on the Macintosh II and the EtherTalk interface card; however, it is envisioned that equivalent interfaces will be provided for other Ethernet interface cards and networking devices.

The Ethernet driver, located in the System file, is named .ENET. If you are developing a driver for use with a slotless device, name the driver .ENET0.

## Write Data Structure

Typically, to send a packet on the network, the driver is called with a write command (see "EWrite Command" this chapter for more information) that contains a pointer to a write data structure (WDS). The WDS contains a series of length and pointer pairs that identify the lengths and memory locations of the packet's components. The WDS for Ethernet is shown in Figure 6-1.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 6-1**
Write Data Structure for Ethernet

The length-pointer pairs tell the driver to gather packet information in the order in which they appear in the WDS. For Ethernet, the first entry in the WDS must point to the 6-byte destination address, which is followed by 6 unused bytes and a 2-byte protocol type. Data may then follow.

- *Note:* When the Ethernet driver transmits the packet, the driver inserts a 6-byte sou rce address to replace the 6 unused bytes.

If you are writing a software driver for transmission of AppleTalk packets on some other network, the first WDS entry may differ from that of Ethernet.

## Protocol Handlers

During a typical read operation, the interface card sends an interrupt to inform the driver that a packet is ready for delivery. The driver responds to this interrupt by reading the Ethernet header into internal driver space and calling a piece of code, known as a *protocol handler*, to process the rest of the packet. The 2-byte protocol type in the header specifies to the driver which protocol handler to call. The protocol handler responds by calling one or both of two driver routines (ReadPacket and ReadRest) to process packet reception.

The Ethernet driver provides a general-purpose default protocol handler for use with the standard read call (see "ERead Command" in this chapter); however, you may decide to write you own protocol handler to process packet reception.

### Writing Protocol-handler Code

After determining how many bytes to read and where to put them, the protocol handler must call one or both of two Ethernet driver routines that perform all low-level manipulations of the card required to read bytes from the network. These two routines are ReadPacket and ReadRest. The protocol handler may call ReadPacket repeatedly to read in the packet piece-by-piece into a number of buffers, as long as it calls ReadRest to read the final piece of the packet. This process is necessary because ReadRest restores state information and checks error conditions. ReadPacket returns an error if the protocol handler attempts to read more bytes than remain in the packet.

When the Ethernet driver passes control to your protocol handler, it passes various parameters and pointers in the processor's registers. Register setup and restrictions are essentially the same as those for ALAP protocol handlers. The Ethernet driver calls the protocol handler as follows:

- A0, A1: reserved for internal use by the driver (handler must preserve until ReadRest is complete)
- A2: free (A2 is not free in an ALAP protocol handler)
- A3: pointer to first byte past data link header bytes (for Ethernet, the byte after the two-byte type field)
- A4: pointer to ReadPacket and ReadRest
- A5: free (until ReadRest is complete)
- D0: free
- D1: number of bytes in packet left to read

D2: free
D3: free

◆ *Note:* ReadRest begins 2 bytes after ReadPacket.

Registers A0, A1, A4, and D1 must be preserved until the protocol handler calls ReadRest. After the protocol handler calls ReadRest, normal interrupt conventions apply. D1 contains the number of bytes remaining to be read in the packet as derived from the packet's length field. D1 can be reduced to indicate pad bytes that will not be read, but should not be changed otherwise.

If the protocol handler is to handle multiple protocol types, the protocol handler should examine the data link header for the protocol-type field to initiate the proper read routine for the incoming packet. Because A3 points to the first byte after the 2-byte protocol type field, the protocol handler can read the type field by using negative offsets from A3. In the case of Ethernet, the 2-byte type field begins at –2(A3), the source address begins at –8(A3), and the destination address is at –14(A3).

## Calling ReadPacket and ReadRest

Your protocol handler can call the Ethernet driver's ReadPacket routine in the following way.

```
JSR     (A4)
```

On entry

D3: number of bytes to be read (word)—must be nonzero
A3: pointer to a buffer to hold the bytes

On exit

D0: modified
D1: number of bytes left to read in packet (word)
D2: preserved
D3: = 0 if requested number of bytes were read; < > 0 if error
A0-A2: preserved
A3: pointer to 1 byte past the last byte read

ReadPacket reads the number of bytes that D3 specifies into the buffer to which A3 points. The number of bytes remaining to read is returned in D1. A3 points to the location to begin reading next time (1 byte following the last byte read).

To read in the rest of the packet, call the Ethernet driver's ReadRest routine in the following way.

```
JSR     2 (A4)
```

On entry

A3: pointer to a buffer to hold the bytes
D3: size of the buffer (word)—can be zero

On exit
D0-D1: modified
D2: preserved
D3: = 0 if requested number of bytes were read
< 0 if packet was -D3 bytes too large to fit in buffer and was truncated
> 0 if D3 bytes were not read (packet is smaller than buffer)
A0-A2: preserved
A3: pointer to 1 byte past the last byte read

ReadRest reads the remaining bytes of the packet into the buffer whose size is given in D3 and whose location is pointed to by A3. The result of the operation returns in D3. If the buffer size that D3 indicates is larger than the packet size, ReadRest does *not* return an error.

---

**Warning**
**Always call ReadRest to read the last part of a packet to avoid a system crash.**

---

If the protocol handler wishes to discard the remaining data before reading the last byte of the packet, it should terminate by calling ReadRest as follows:

```
MOVEQ     #0,D3      ;byte count of 0
JSR       2(A4)      ;call ReadRest
RTS
```

In all cases, the protocol handler should end with an RTS, even if the driver returns an error. If the driver returns an error from a ReadPacket call, the protocol handler must quit via an RTS without calling ReadRest at all. Upon return from ReadRest and ReadPacket, the zero (z) bit in the command control register will be set if an error did *not* occur. If an error occurs, the zero bit is not set.

## Opening the Ethernet Driver

On a Macintosh II, use the Device Manager to make a PBOpen call to open the Ethernet driver. However, you will have to obtain certain field values, such as the card's slot number, before making this call. The Slot Manager may be used to obtain these values.

## Slot Manager sNextsRsrc Trap Macro

There are six slots in the Macintosh II, ranging from $09 to $0E. Built-in devices use slot zero. Because these slots may contain interface cards other that EtherTalk cards, your code must identify the type of card in each slot. One method of doing this is to use the Slot Manager sNextsRsrc trap macro. This function is defined as follows:

| | | |
|---|---|---|
| Required Parameters | <—> | spSlot |
| | <—> | spId |
| | <— | spsPointer |
| | <— | spRefNum |
| | <— | spIOReserved |
| | <—> | spExtDev |
| | <— | spCategory |
| | <— | spCType |
| | <— | spDrvrSw |
| | <— | spDrvrHw |

If you supply a 0 for the sNextsRsrc fields spSlot, spID, and spExtDev, this routine returns the spId, spSlot, spCategory, and spType values in addition to other information for each card installed. The routine starts at slot $09 and continues to slot $0E and returns a non-fatal error report to indicate routine completion. By matching the spCategory and spType fields to the sResource Type format for the EtherTalk interface card, your code can identify which slots contain EtherTalk cards. The sResource Type format for the EtherTalk interface card identifies the spCategory field as *CatNetwork* and the spType field as *TypEthernet*.

◆ *Note:* More information about the sResource Type formats is contained in MPW, version 2.0. Refer to Inside Macintosh, Chapter 24, Volume V for more information on the Slot Manager.

## Device Manager PBOpen Call

Once you have obtained the slot number (spSlot) and the sResource Identification number (spId), you may use the Device Manger to make a PBOpen call. The PBOpen call requires that you supply, in addition to other information, the driver name (.ENET) and the ioSlot and ioId parameters as this portion of the parameter block shows:

| | | | |
|---|---|---|---|
| —> | 34 | ioSlot | byte |
| —> | 35 | ioId | byte |

For the EtherTalk card, the ioSlot parameter contains the slot number, obtained as spSlot, for the EtherTalk card to be opened, in the range of $09 to $0E. The ioId parameter contains the sResource ID, obtained as spId. Be sure to set the immediate-bit in the trap word when making the PBOpen call. Refer to *Inside Macintosh*, Chapter 23, Volume V for additional information about opening slot devices.

The driver opens in the AppleTalk mode. This mode limits the size of a packet sent by the driver to 768 bytes, which is more than sufficient to encapsulate a maximum-size Ethernet-AppleTalk packet of 617 bytes. In this mode, the driver can allocate more space in the buffer pool for packet reception. If the packets for your application require more than 768 bytes, you may change to General mode to transmit any valid Ethernet packet up to 1514 bytes in length.

**<<Network Engineering to provide sample code for opening the driver>>**

## Making Commands to the Ethernet Driver

Once the Ethernet driver opens, a series of Device Manager control calls to the driver are made to control packet transmission and reception. The calling code passes command arguments in the queue element starting at CSParam. Refer to Chapter 6 of *Inside Macintosh*, Volume II, for more information about making control calls.

### The EWrite Command

Use the Device Manager to make the EWrite control call to write a packet out on Ethernet. The only argument is a pointer that identifies the location of the write data structure used to send the packet on the network.

Parameter block

| | | | |
|---|---|---|---|
| —>26 | csCode | word | {always EWrite} |
| —>30 | EWdsPointer | pointer | {write data structure} |

**<<Result codes for all commands to be supplied by Network Engineering>>**

### The EAttachPH Command

Make the EAttachPH command to attach a protocol handler to the driver. Arguments are a 2-byte protocol type and a protocol-handler address. If the protocol-handler address is 0, a default protocol handler is attached to the Ethernet driver. The default protocol handler is for use with the ERead call (see "ERead Command" this chapter).

Parameter block

| | | | |
|---|---|---|---|
| —>26 | csCode | word | {always EAttachPH} |

| | | | |
|---|---|---|---|
| —>28 | EprotType | 2 bytes | {Ethernet protocol type} |
| —>30 | Ehandler | pointer | {protocol handler} |

EAttachPH adds the protocol handler pointed to by Ehandler to the node's protocol table. EprotType specifies what kind of packet the protocol handler can service. After EAttachPH is called, the protocol handler is called for each incoming packet whose Ethernet protocol type equals EprotType.

◆ *Note:* To attach or detach a protocol handler for IEEE 802.3, which uses protocol types 0 through $5DC, specify protocol type zero.

## The EDetachPH Command

Makes the EDetachPH command to detach a protocol handler from the driver.

Parameter block

| | | | |
|---|---|---|---|
| —>26 | csCode | word | {always EDetachPH} |
| —>28 | EprotType | 2 bytes | {Ethernet protocol type} |

The command removes the protocol type and corresponding protocol handler from the protocol table.

## The ERead Command

Make the ERead call only to read in a packet after an EAttachPH with a zero-handler address is issued for the protocol indicated in this command. ERead takes as arguments the protocol type, buffer pointer, and buffer size. The ERead call places the entire packet, including the header, into the buffer. After the read, the call returns the actual size of the packet. If the packet is too large to fit into the buffer, the call places as much of the packet as it can into the buffer and returns an error. The driver dequeues the ERead call from the system queue, so more than one ERead call can be active concurrently.

Parameter block

| | | | |
|---|---|---|---|
| 26 | —> | csCode | {always ERead} |
| 28 | —> | EProtType | {protocol type} |
| 30 | —> | EBuffPtr | {buffer into which packet is read} |
| 34 | —> | EBuffSize | {buffer size} |
| 36 | <— | EDataSize | {actual number of bytes read} |



CONFIDENTIAL

## The ERdCancel Command

The ERdCancel command cancels a particular ERead call. The only argument is the queue- element pointer of the ERead call to cancel. If the ERead call is active, the ERdCancel call returns an error.

Parameter block

| | | | |
|---|---|---|---|
| 26 | —> | csCode | {always ERdCancel} |
| 30 | —> | EKillQEl | {queue element pointer to cancel} |

## The EGetInfo Command

The EGetInfo command obtains driver information and takes arguments of a buffer pointer and size. This call returns, in the first 6 bytes of the buffer, the Ethernet address for the node on which the driver is installed.

Parameter block

| | | | |
|---|---|---|---|
| 26 | —> | csCode | {always EGetInfo} |
| 30 | —> | EBuffPtr | {buffer pointer} |
| 34 | —> | EBuffSize | {buffer size} |

With the EtherTalk driver installed on a Macintosh II, the EGetInfo call returns 12 additional bytes as follows:

Bytes 07–10 = number of buffer overwrites on receive

Bytes 11–14 = number of time-outs on transmit

Bytes 15–18 = number of packets received that contain an incorrect address

## The ESetGeneral Command

The ESetGeneral command changes the driver from AppleTalk mode to General mode. There are no arguments. There is no command to change the driver from General mode to AppleTalk mode. Changing the driver's mode may involve a hardware reset, and could cause loss of an incoming packet.

Parameter block

| | | | |
|---|---|---|---|
| 26 | —> | csCode | {always ESetGeneral} |

# Chapter 7

# The EtherTalk Interface Card

This chapter overviews the operation of the EtherTalk Interface card and generally explains each major component on the card. In addition to this information, this chapter identifies the address assignments for local menory and gives the address assignments for the network interface controller (NIC) register.

## About the EtherTalk Card

The EtherTalk Interface card installs in any NuBus slot on a Macintosh II computer. The card interfaces the Ethernet driver to the Ethernet cabling system to enable packet transmission and reception among EtherTalk nodes. The EtherTalk Interface card may also function in Apple Computer's A/UX environment, transporting transmission control protocol/internet protocol (TCP/IP) packet information. Please note that the A/UX operating system does *not* support EtherTalk software.

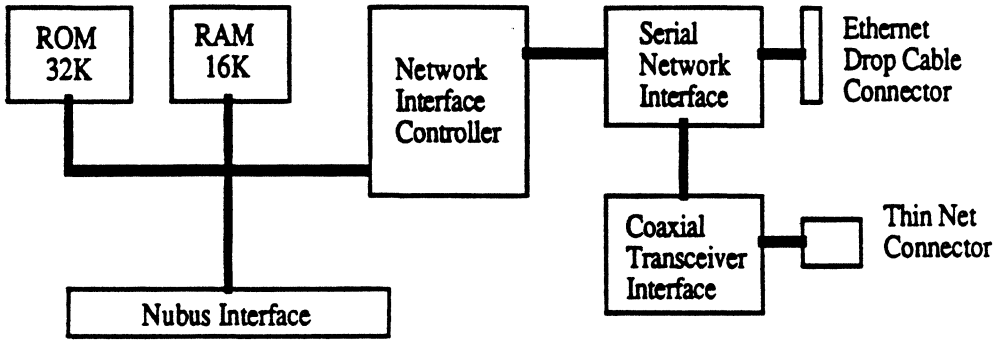
A detailed discussion of the A/UX operating system or device drivers is beyond the scope of this document. Please refer to the Apple publications *Building A/UX Device Drivers, A/UX Programmer's Reference,* and *A/UX Networking Applications Programming* for more information about TCP/IP, A/UX, and device drivers.

<<Alan, this chapter is lifted from the Schlitz Preliminary Note. The information presented here seems to be a little weak. Is there anything else that you think needs to be here? How about a more detailed explanation of drop cable and Thin Net connections and give some programming application information on how to use the local memory addresses and NIC register addresses? >>

## EtherTalk Card Hardware Description

The EtherTalk Interface Card is a non-intelligent Ethernet adapter for the Macintosh II computer. The card uses a local-area-network (LAN) chipset from National Semiconductor Corporation. The three LAN chips are the NIC, Serial Network Interface (SNI), and a Coaxial Transceiver Interface (CTI). The card has 16K of dual-ported Random Access Memory (RAM) and 32K of Read Only Memory (ROM). The local memory allows back-to-back packet reception with multipacket buffering.

Figure 7-1 shows the architecture of the Ethernet Interface card.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 7-1**
EtherTalk Interface Card Architecture

The EtherTalk card uses the Apple's implementation of the Nubus interface. More information on the Apple's implementation of Nubus can be found in the Apple Computer publication *Macintosh II and Macintosh SE Cards and Drivers* available from Apple Programers and Developers Association (APDA).

The CTI chip is used as a coaxial line driver and receiver for Thin Net LANs. The CTI is not used when attaching to an Ethernet backbone cable by means of an external transceiver <<what external transceiver?>> The selection is made with a jumper on the EtherTalk card <<what selection?>>

The SNI chip provides the encoding and decoding functions <<of what?>>for Ethernet or Thin Net LANs. The SNI also provides a collision signal translator and a diagnostic loopback circuit. <<more info here?>>

The NIC chip is the heart of the LAN chipset. The NIC performs all Media Access Control (MAC) layer functions for transmission and reception of Ethernet packets. The NIC provides buffer management that supervises storage of received packets in the local memory. During packet transmission, the NIC generates and appends the preamble and sync byte to the transmitted packet. Also, the NIC will optionally compute and append Cyclic Redundancy Check (CRC) bytes. During reception, the NIC decodes and filters addresses and performs CRCs. More programming information about the NIC can be found in the National Semiconductor specification document titled *DP8390/NS32490 Network Interface Controller.*

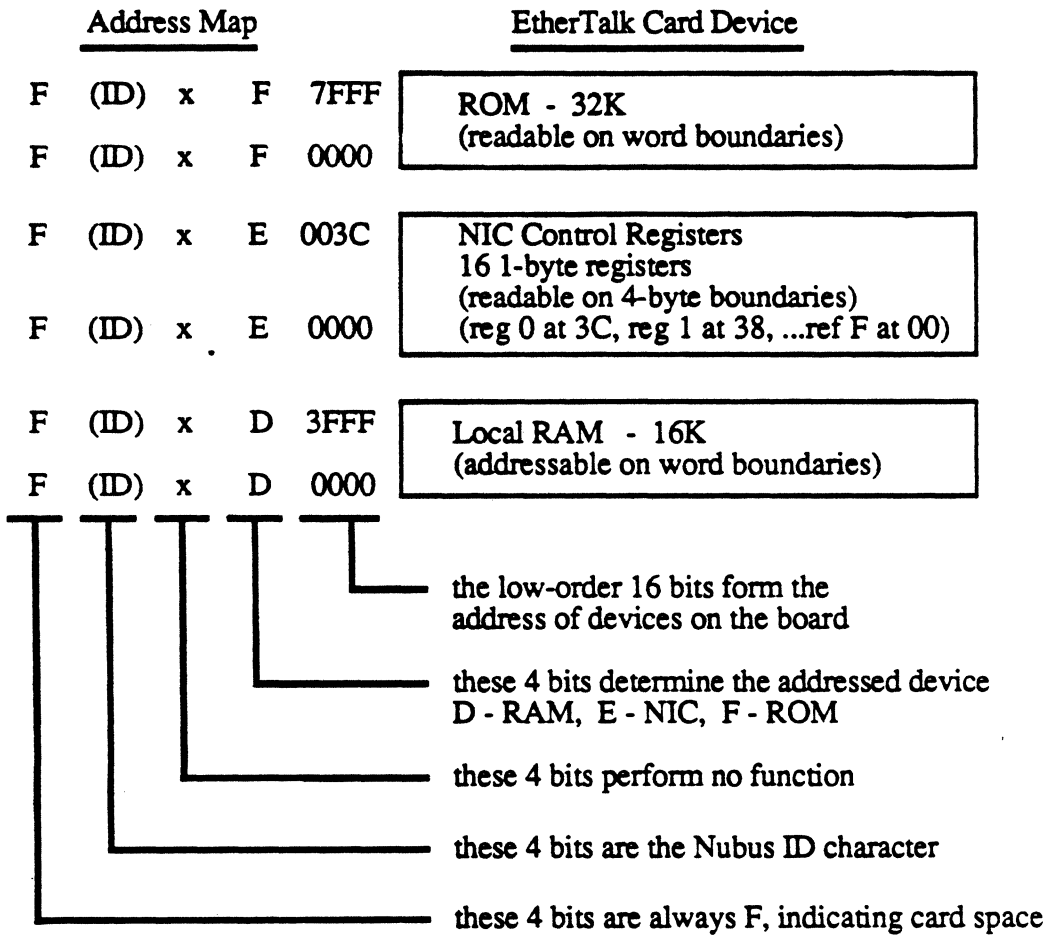
## Local Memory

The local memory consists of 16K of static RAM segmented into *transmit* and *receive buffers* by setting registers in the NIC. The segments are further divided into 256-byte *pages*. Some number of pages (under driver control) is used for a transmit buffer. The remaining pages are used for a receive ring buffer. Page Start and Page Stop registers establish a buffer that forms a continuous address space. As the last address is reached (set up by the Page Stop register), the next memory location wraps around to the start of the buffer (set up by the Page Start register) to form the receive ring buffer.

The local ROM memory is 32K in size and contains the Ethernet address and Nubus card slot information. Direct access of the ROM is not usually necessary because these services are provided by the slot library available in A/UX.

## Address Assignments

Figure 7-2 shows the address map of devices on the EtherTalk card.




MSC NNNN
ART: NN x 17 pi
20.5 pi text to FN b/b

**Figure 7-2**
Address Assignments

## NIC Register Addresses

NIC registers are divided into three pages. The content of the highest-order bits in the Command register (PS0 and PS1) defines which page of registers is being read from and written to for addresses E0000 through E003C. Page 0 registers are those registers commonly accessed during normal operation of the NIC. Page 1 registers are accessed during the initialization process. Page 2 registers should only be accessed for diagnostic purposes and should not be modified during normal operation. For complete definition of the register terms in the tables, consult the National Semiconductor Corporation publication *DP8390/NS32490 Network Interface Controller.*

The following tables show the registers used for programming the NIC. Table 1 displays the NIC Page 0 register addresses on the EtherTalk card.

**Table 7-1**
Page 0 Address Assignments (PS1=0, PS0=0)

| Address | Page 0 Read | Page 0 Write |
|---|---|---|
| E003C | Command Register | Command Register |
| E0038 | Current Local DMA Address (CLDA) 0 | Page Start Register |
| E0034 | Current Local DMA Address (CLDA) 1 | Page Stop Register |
| E0030 | Boundary Register | Boundary Register |
| E002C | Transmit Status Register | Transmit Page Start Register |
| E0028 | Number of Collisions Register | Transmit Byte Count Register (TBCR) 0 |
| E0024 | First In First Out (FIFO) Register | Transmit Byte Count Register (TBCR) 1 |
| E0020 | Interrupt Status Register | Interrupt Status Register |
| E001C | Current Remote Data Address (CRDA) 0 | Remote Start Address Register (RSAR) 0 |
| E0018 | Current Remote Data Address (CRDA) 1 | Remote Start Address Register (RSAR) 1 |
| E0014 | Reserved | Remote Byte Count Register (RBCR) 0 |
| E0010 | Reserved | Remote Byte Count Register (RCBR) 1 |
| E000C | Receive Status Register | Receive Configuration Register |

| | | |
|---|---|---|
| E0008 | Counter (CNTR) 0 | Transmit Configuration Register |
| E0004 | Counter (CNTR) 1 | Data Configuration Register |
| E0000 | Counter (CNTR) 2 | Interrupt Mask Register |

The byte counts in Transmit Byte Count registers 0 and 1 are combined to create a single count. The byte counts in Remote Byte Count registers 0 and 1 are also combined. Counter 0 is used for frame alignment errors, counter 1 for CRC errors, and counter 2 for missed packet errors.

Table 2 displays the NIC Page 1 register addresses on the EtherTalk card.

**Table 7-2**
Page 1 Address Assignments (PS1=0, PS0=1)

| Address | Page 0 Read | Page 0 Write |
|---|---|---|
| E003C | Command Register | Command Register |
| E0038 | Physical Address Register (PAR) 0 | Physical Address Register (PAR) 0 |
| E0034 | Physical Address Register (PAR) 1 | Physical Address Register (PAR) 1 |
| E0030 | Physical Address Register (PAR) 2 | Physical Address Register (PAR) 2 |
| E002C | Physical Address Register (PAR) 3 | Physical Address Register (PAR) 3 |
| E0028 | Physical Address Register (PAR) 4 | Physical Address Register (PAR) 4 |
| E0024 | Physical Address Register (PAR) 5 | Physical Address Register (PAR) 5 |
| E0020 | Current Point (CURR) | Current Point (CURR) |
| E001C | Multicast Address Register (MAR) 0 | Multicast Address Register (MAR) 0 |
| E0018 | Multicast Address Register (MAR) 1 | Multicast Address Register (MAR) 1 |
| E0014 | Multicast Address Register (MAR) 2 | Multicast Address Register (MAR) 2 |
| E0010 | Multicast Address Register (MAR) 3 | Multicast Address Register (MAR) 3 |
| E000C | Multicast Address Register (MAR) 4 | Multicast Address Register (MAR) 4 |
| E0008 | Multicast Address Register (MAR) 5 | Multicast Address Register (MAR) 5 |
| E0004 | Multicast Address Register (MAR) 6 | Multicast Address Register (MAR) 6 |
| E0000 | Multicast Address Register (MAR) 7 | Multicast Address Register (MAR) 7 |

<<Change for EtherTalk>>The operating system reads the EtherTalk card ROM and installs 6 bytes into the Physical Address registers 0 through 5. The Current Point is a page where a packet is currently being received; it is used to detect packet reception. The Multicast Address registers are initialized to 0XFF because A/UX does not support multicasting.

# Appendix A

# EtherTalk Components

## Component List

Table A-1 lists the location, resource type, and description of each EtherTalk software component.

**Table A-1**
EtherTalk Components

| Location | Type | ID | Name | Description |
|---|---|---|---|---|
| System File | DRVR | 127 | .ENET | EtherNet driver for Macintosh II EtherTalk interface card. |
| | ALRT | -4031 | | Alerts and associated dialog item lists used at boot time to indicate an error ocurred while installing the altenate AppleTalk selection. ALRTs and DITLs must be installed with the LAP Manager INIT Resource. |
| | ALRT | -4032 | | |
| | • | | | |
| | DITL | -4031 | | |
| | DITL | -4032 | | |
| | INIT | 18 | | LAP Manager INIT resource. Contains LAP Manager code plus other code to install the alternate AppleTalk selection at startup time. |
| System Folder | | | Network | Network 'cdev' file. Contains code to implement one or more alternate AppleTalk Selections. |
| | | | EtherTalk | EtherTalk 'adev' file. Contains code to implement one or more alternate AppleTalk selections. |

# Ethernet Driver Equates

Table A-2 lists the equates for the .ENET driver.

**Table A-2**
Ethernet Driver Equates

| Group and Name | Equate | Comment |
|---|---|---|
| **Control codes** | | |
| ESetGeneral | EQU 253 | Set General mode |
| EGetInfo | EQU 252 | Get info |
| ERdCancel | EQU 251 | Cancel read |
| ERead | EQU 250 | Read |
| EWrite | EQU 249 | Write |
| EDetachPH | EQU 248 | Detach protocol handler |
| EAttachPH | EQU 247 | Attach protocol handler |
| FirstENET | EQU EAttachPH | First ENET command |
| LastENET | EQU ESetGeneral | Last ENET command |
| **ENET queue element standard structure—arguments passed in the CSParam area** | | |
| EProtType | EQU CSParam | Offset to protocol type code |
| EHandler | EQU EProtType+2 | Offset to protocol handler |
| EWDSPointer | EQU EHandler | WDS pointer (EWrite) |
| EBuffPtr | EQU EHandler | Buffer pointer (ERead, EGetInfo) |
| EKillQEl | EQU EHandler | QEl pointer (EReadCancel) |
| EBuffSize | EQU EBuffPtr+4 | Buffer size (ERead, EGetInfo) |
| EDataSize | EQU EBuffSize+2 | Actual data size (ERead) |
| **Equates for the Ethernet packet header** | | |
| EDestAddr | EQU 0 | Offset to destination address |
| ESrcAddr | EQU 6 | Offset to source address |
| EType | EQU 12 | Offset to data link type |
| EHdrSize | EQU 14 | Ethernet header size |
| EMinDataSz | EQU 46 | Minimum data size |
| EMaxDataSz | EQU 1500 | Maximum data size |
| EAddrSz | EQU 6 | Size of an Ethernet node address |
| MAddrSz | EQU 8 | Size of an Ethernet multicast address |

## LAP Manager Equates

Table A-3 lists the equates for the LAP Manager.

**Table A-3**
LAP Manager Equates

| Group and Name | Equate | Comment |
|---|---|---|
| LAP Manager call codes passed in D0 (call at [ATalkHk2] + 2) | | |
| LRdDispatch | EQU 1 | Dispatch to protocol handler |
| LWrtInsert | EQU 2 | Insert in LAPWrite hook |
| LWrtRemove | EQU 3 | Remove from LAPWrite hook |
| LWrtGet | EQU 4 | Get code in LAPWrite hook |
| LSetInUse | EQU 5 | Set address-in-use flag |
| LGetSelfSend | EQU 6 | Get value of self-send flag |
| LAARPAttach | EQU 7 | Attach an AARP listener |
| LAARPDetach | EQU 8 | Detach an AARP listener |
| LGetATalkInfo | EQU 9 | Get AppleTalk info |
| Flag bits passed in D1 on LWrtInsert | | |
| LWSelfSend | EQU 7 | ADEV handles self-send |
| LWEnableSCC | EQU 6 | Do not disable SCC |
| LWSrvrWks | EQU 5 | Honor server/wks bit |
| 'atlk' resource call codes passed in D0 (call at atlk + 2) | | |
| AInstall | EQU 1 | Installation |
| AShutdown | EQU 2 | Shutdown |
| atlkCall | EQU2 | Offset at which to make calls |
| ADEV file call code passed in D0 (call at ADEV start) | | |
| GetADEV | EQU 101 | Get next ADEV |
| SelectADEV | EQU 102 | Select ADEV |
| Low-memory equates | | |
| LAPMgrPtr | EQU $B18 | This points to our start |
| LAPMgrCall | EQU 2 | Offset to make LAP Manager calls |
| ATalkPRAM | EQU $E0 | Start of our parameter RAM |
| LAPMgrByte | EQU $60 | Value of byte pointed to by LAPMgrPtr |
| Resource ID | | |
| adevBaseID | EQU -4032 | Base resource ID for ADEVs |

## AARP Equates

Table A-4 lists the equates for AARP.

**Table A-4**
AARP Equates

| Group od Name | Equate | Comment |
|---|---|---|
| AARP protocol type: | | |
| AARP | EQU $80F3 | |
| Offsets in packet | | |
| AAHardware | EQU 0 | Hardware type |
| AAProtocol | EQU AAHardware+2 | Protocol type |
| AAHLength | EQU AAProtocol+2 | Hardware length |
| AAPLength | EQU AAHLength+1 | Protocol length |
| AACommand | EQU AAPLength+1 | AARP command |
| AAData | EQU AACommand+2 | Data start |
| AARP commands | | |
| AARPReq | EQU 1 | Request |
| AARPResp | EQU 2 | Response |
| AARPProbe | EQU 3 | Probe |
| EtherTalk specifics | | |
| H_Ethernet | EQU1 | Hardware type for Ethernet |
| HL_Ethernet | EQU6 | Ethernet address length |
| P_AppleTalk | EQU $809B | Protocol type for AppleTalk |
| PL_AppleTalk | EQU 4 | AppleTalk address length |
| AAESrcPhys | EQU AAData | Source hardware address offset |
| AAESrcLog | EQU AAESrcPhys+ HL_Ethernet | Source protocol address |
| AAEDstPhys | EQU AAESrcLog+ PL_AppleTalk | Destination hardware address |
| AAEDstLog | EQU AAEDstPhys+ HL_Ethernet | Destination protocol address |
| AAEEnd | EQU AAEDstLog+ L_AppleTalk | End of packet |
| Retransmission equates | | |
| APrbTicks | EQU 2 | Number of ticks between probes |
| AReqTicks | EQU 2 | Number of ticks between requests |
| AReqTries | EQU 6 | Number of tries on requests |

## ADEV File Boilerplate

**<<To be supplied by Developer Tech. Support>>**