Macintosh™

EtherTalk and Alternate AppleTalk Connections Reference

Working Draft 4 - June 27, 1988

Communications & Networking
Apple Technical Publications

Engineering Part No. 6588279

Contents

Figures and tables

Preface

This reference is intended to be used by Apple® software developers who wish to develop an alternate AppleTalk® connection or an Ethernet application in conjunction with the operating system of the Macintosh® computer. 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.

What this reference contains

This reference provides you with an overview of Apple's EtherTalk™ software, as well as a detailed description of each software component. This reference also discusses call definitions, register usage, and call applications.

Suggested reading

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

- *Inside AppleTalk* (Apple Programmers and Developers Association)
- *Inside Macintosh*, Volume II, Chapter 6 (Apple Computer, Inc.)
- *Inside Macintosh*, Volume II, Chapter 10 (Apple Computer, Inc.)
- *Inside Macintosh*, Volume V, Chapter 23 (Apple Computer, Inc.)
- *Inside Macintosh*, Volume V, Chapter 24 (Apple Computer, Inc.)
- *Inside Macintosh*, Volume V, Chapter 28 (Apple Computer, Inc.)
- *Ethernet Blue Book* (Xerox Corporation)
- *EtherTalk User Guide* (Apple Computer, Inc.)

Possible applications

There are many possible applications that you may wish to develop. For example, you may want to create your own alternate AppleTalk connection 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.

Visual clues and conventions

Look for these visual clues throughout the reference:

❖ *Note:* Notes like this contain supplementary information.

Terms in **boldface** type are defined in the glossary.

A special typeface is used to indicate lines of code:

```
It looks like this or this.
```

Values represented in hexadecimal are preceded by a dollar sign ($).

The symbol —> indicates that a value is passed out of a parameter block. This symbol also indicates that the contents of a register contain an address pointer.

The symbol <— indicates that a value is returned to a parameter block.

The symbol <—> indicates that a value is passed out of a parameter block and another value is returned to the parameter block.

The term *alternate AppleTalk connection* refers to a network selection other than LocalTalk™.

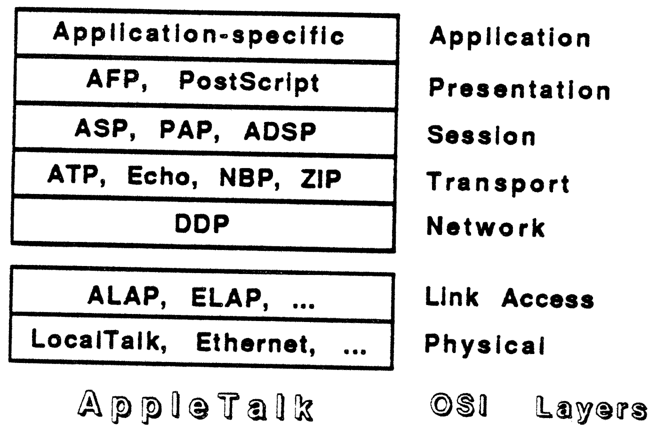
Chapter 1

Introduction

The name *AppleTalk* refers to a *system* of hardware and software components that transfer information when connected by a physical medium. LocalTalk, EtherTalk, AppleShare™, and LaserShare™ are all components of the AppleTalk system.

Apple Computer, Inc., has 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 (such as Physical and Data Link) of the International Standards Organization-Open Systems Interconnection (ISO-OSI) reference model. Figure 1-1 illustrates the AppleTalk protocol model.

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



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

Figure 1-1
AppleTalk protocol model

AppleTalk connections

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

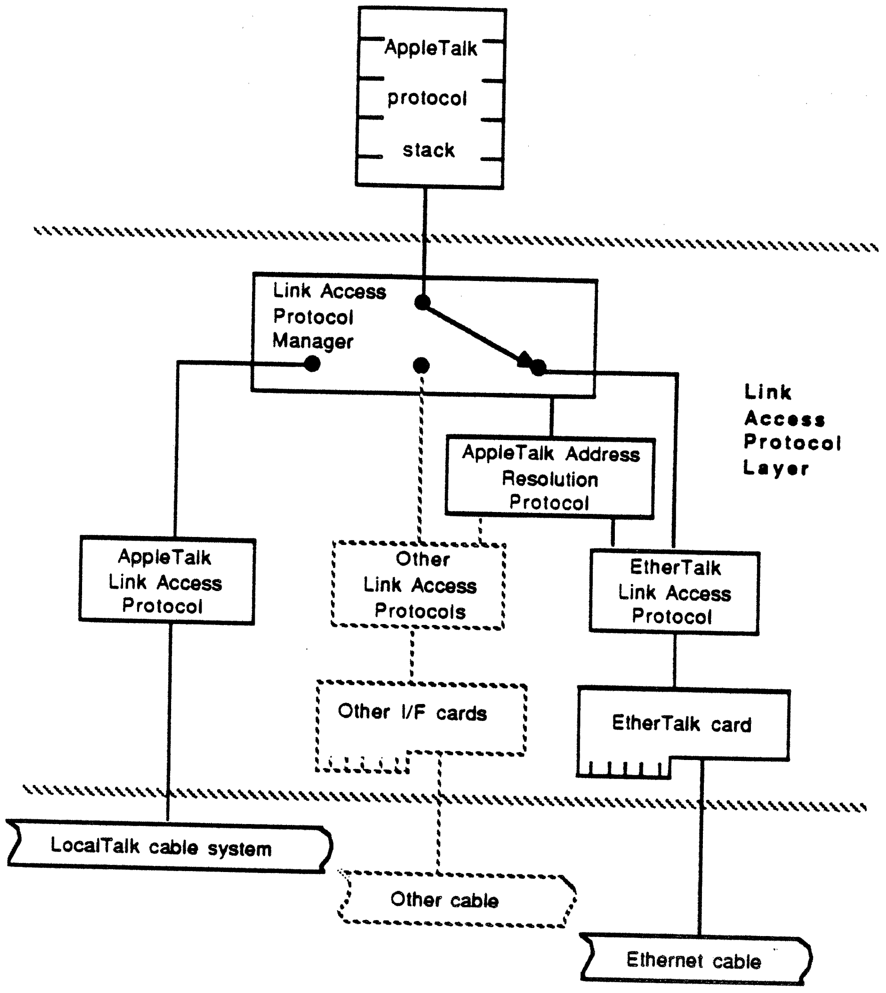
Before the development of EtherTalk, the only option available to the user was to transfer information over the LocalTalk cable system or its equivalent by using the AppleTalk Link Access Protocol (ALAP) to perform node-to-node delivery of information. While this option was sufficient for many situations, the Macintosh could only transfer information on LocalTalk cables.

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 LocalTalk connection, Ethernet, or any other LAPs that support additional networks. Figure 1-2 shows the way that LocalTalk, EtherTalk, or other similar types of connections interact with the AppleTalk system from a Macintosh II computer.

Currently, the Macintosh has only one active connection at any given time. However, future developments may require the Macintosh to have two or more connections that are concurrently active. This would be useful for bridging operations. The LAP Manager provides a mechanism for supporting multiple simultaneously active AppleTalk connections.

LAP functions

ALAP assigns a unique identification number to each device, or node, on the LocalTalk cable system. 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 for LocalTalk 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—the 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 reference discusses AARP and driver-level ELAP in more detail in later chapters.

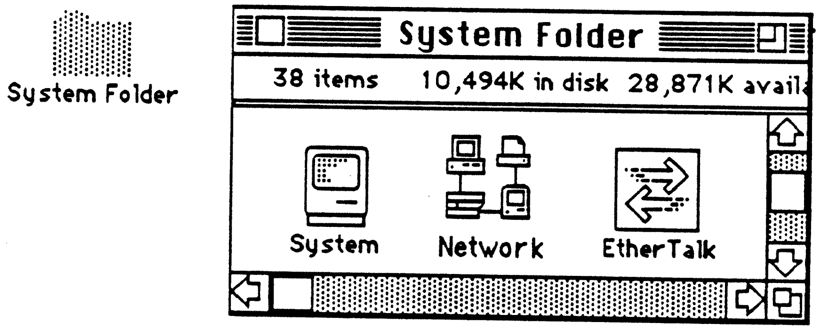

AppleTalk
protocol
stack
Link Access Protocol Manager
AppleTalk Address Resolution Protocol
Link Access Protocol Layer
AppleTalk Link Access Protocol
Other Link Access Protocols
EtherTalk Link Access Protocol
Other I/F cards
EtherTalk card
LocalTalk cable system
Other cable
Ethernet cable

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

Figure 1-2
AppleTalk connections

Using EtherTalk

EtherTalk software is designed to operate with the Macintosh Operating System (O/S) 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 it contains a compatible Ethernet interface card and driver. As shown in Figure 1-3, the EtherTalk software includes in the System Folder, a Network device file and an EtherTalk defice file along with the System file and its other device files (such as Mouse, Keyboard, etc.). The EtherTalk software also contains a Utitities folder with an installer program and its related files.



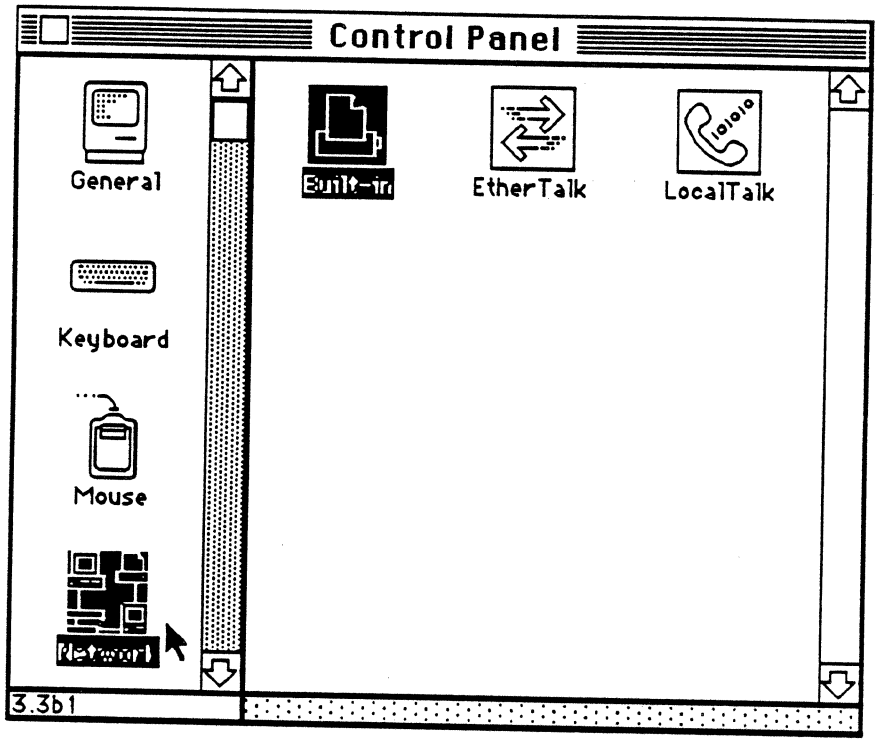
MSC NNNN
ART: NN x 8.5 pi
12 pi text to FN b/b
Figure 1-3
The System and device files for EtherTalk

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. The EtherTalk icon represents only one type of connection; however there may be more than one EtherTalk connection supported by the system. The Built-in icon represents a LocalTalk connection on the printer port of the Macintosh II. The LocalTalk icon represents a LocalTalk connection on the modem port of the Macintosh II.

The active connection is highlighted on the Network Control Panel.

Figure 1-4 shows a sample Network Control Panel. In the figure, the LocalTalk connection through the printer port is active.

❖ *Note:* The active button for AppleTalk must be active in the Chooser for any AppleTalk connection to operate.



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

Figure 1-4
The Control Panel display for selecting AppleTalk connections

EtherTalk software contains these components in addition to the Control Panel software:

- ❑ the Ethernet Driver, which is the interface to the Ethernet card
- ❑ the LAP Manager, which standardizes interaction with AppleTalk drivers
- ❑ 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

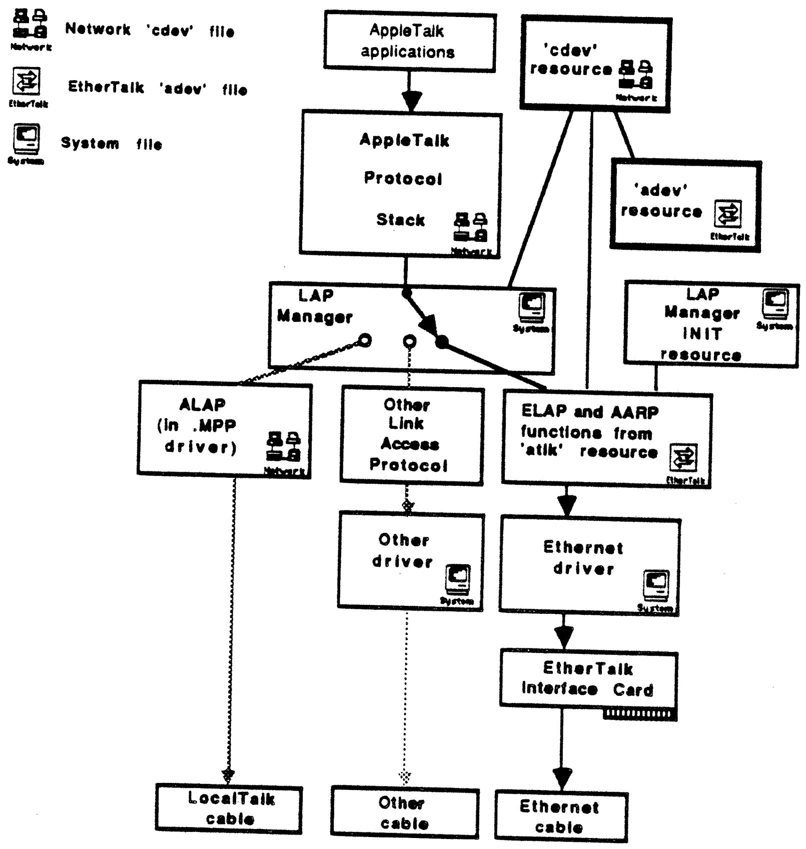
Chapter 2

EtherTalk Overview

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

An EtherTalk block diagram

Figure 2-1 shows all EtherTalk components and the way that these components relate to the AppleTalk environment. In the figure, the icon shown in each box indicates the file in which the specified component resides.



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

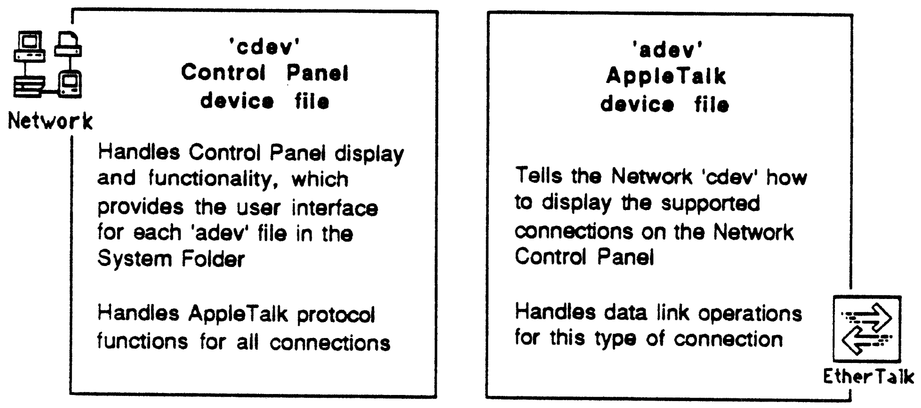
Figure 2-1
The relationship of the EtherTalk components

Device files 'cdev' and 'adev'

The Control Panel device files (file type 'cdev') and the AppleTalk device files (file type 'adev') both reside in the System Folder. EtherTalk software contains the Network 'cdev' file and the EtherTalk 'adev' file. These device files work together to handle the user interface (through the Control Panel) and to support the communications functions for the Macintosh II. Figure 2-2 illustrates the relationship between the Network 'cdev' and EtherTalk 'adev' files.

Each type of AppleTalk connection (other than LocalTalk) must have its own 'adev' file; however, an 'adev' file may support more than one connection of the same type. For example, the EtherTalk 'adev' supports an EtherTalk connection for each EtherTalk Interface Card installed in the Macintosh II.

❖ *Note:* There is no BuiltIn 'adev' file. The code that supports LocalTalk connections is part of the Network 'cdev' file.


Network
'cdev'
Control Panel
device file
Handles Control Panel display and functionality, which provides the user interface for each 'adev' file in the System Folder
Handles AppleTalk protocol functions for all connections
'adev'
AppleTalk
device file
Tells the Network 'cdev' how to display the supported connections on the Network Control Panel
Handles data link operations for this type of connection
EtherTalk

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

Figure 2-2
The Network 'cdev' and the EtherTalk 'adev' files

Control Panel device file

Control Panel device files contain various resources that communicate machine options in some form (by using 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. Each 'cdev' file has a unique icon and string associated with it.

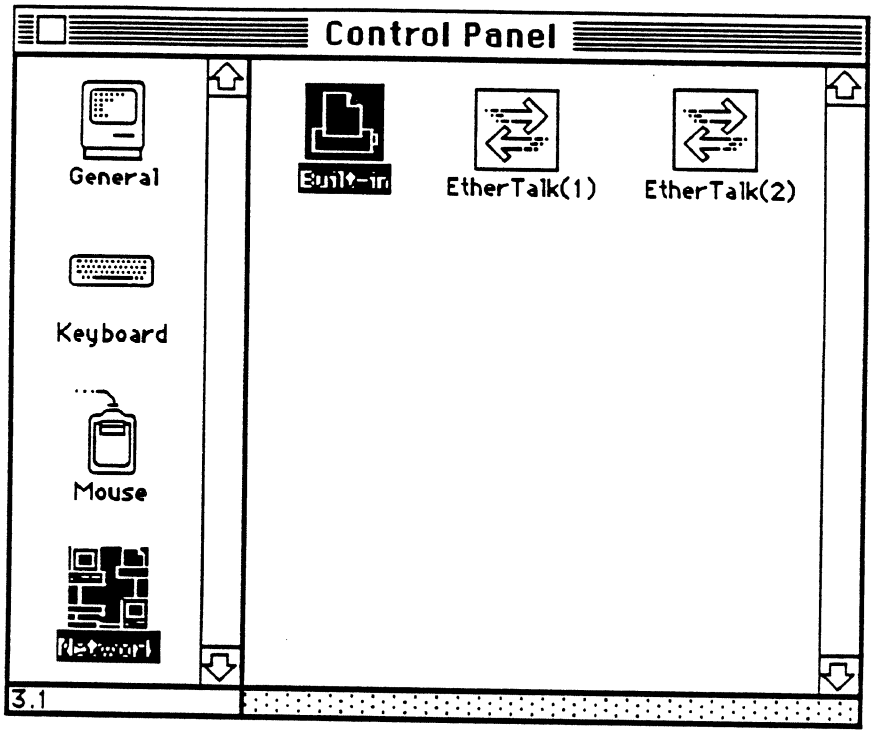
EtherTalk software contains the Network 'cdev' file, located in the System Folder. The Network 'cdev' provides the interface that allows the user to choose which AppleTalk connection is active. The Network 'cdev' file contains various resources to display the available AppleTalk connections and to communicate selection information to the system. The Network icon is shown in Figure 1-3 as it appears in the System Folder and in Figure 2-3 as it appears when it is selected on the Control Panel.

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

When the user chooses the Control Panel from the Menu, the Control Panel scans the System Folder for files of type 'cdev'. Upon finding a 'cdev' file, the Control Panel adds the 'cdev' file's icon and string to a list on the left side of the Control Panel. After adding all of the 'cdev' file icons to the list, the Control Panel highlights the icon of the General 'cdev' file and constructs its control information in the window's content area. At this point, the user may select items from the General Control Panel or may scroll through the list of 'cdev' file icons and may select any one of the icons in the list.

When the Network icon on the Control Panel is selected, the Network 'cdev' file displays a series of icons to represent each available AppleTalk connection. In most cases, the icon representing the active AppleTalk connection is highlighted.

Figure 2-3 shows a Control Panel display that may appear after the user selects the Network icon. In the figure, the Built-in AppleTalk connection is active and two alternate AppleTalk connections, EtherTalk(1) and EtherTalk(2), are available to the user. (The two EtherTalk icons shown in the figure represent two EtherTalk cards installed in the Macintosh II).



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

Figure 2-3
A sample Network Control Panel display

❖ *Note:* The term *Network Control Panel* refers to the Control Panel display when the Network icon is selected.

When the user selects the Network icon, the Network 'cdev' scans the System Folder for all files of type 'adev'. Like the 'cdev' files, each 'adev' file has an icon and string associated with it. Because an 'adev' file supports one type of connection, but may support more than one connection, the Network 'cdev' accesses each 'adev' file to obtain information about the number of identical icons to display and the strings for each icon.

For example, if two EtherTalk cards are installed in the Macintosh II, the EtherTalk 'adev' file, which supports both cards, instructs the Network 'cdev' to display two EtherTalk icons and to 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 Network Control Panel, the user may select one of the AppleTalk icons. When the user makes a new selection, the Network 'cdev' highlights this icon and performs various operations to inform the system of the newly selected AppleTalk connection.

AppleTalk device file

The construction of an 'adev' file is similar to that of a 'cdev' file. For each type of AppleTalk connection, 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, respectively, that the Network 'cdev' file displays in the content area of the Control Panel to represent a particular AppleTalk connection. In addition, if the 'adev' file contains the 'BNDL', 'FREF', and an owner resource, and if the 'adev' file has its bundle bit set, the 'ICN#' will appear in the Finder. Refer to *Inside Macintosh,* Volume 1, for more information about the 'ICN#', 'STR ', 'BNDL', and 'FREF' resources, and the owner resources.

The 'adev' and 'atlk' resources

The 'adev' and 'atlk' resources are pieces of stand-alone code that reside in the 'adev' file. The 'adev' resource is responsible for handling all Control Panel interactions 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 connection, and removes the 'adev' resource as the Network 'cdev' requires.

The 'atlk' resource contains the actual implementation code for the supported AppleTalk connections. The Network 'cdev' loads the 'atlk' resource into the system heap, calls the 'atlk' resource for initialization and installation, and detaches the 'atlk' resource. Because the Network 'cdev' detaches the 'atlk' resource, the currently selected AppleTalk connection remains in effect after the Control Panel interactions are completed and the 'adev' file is closed.

The LAP Manager INIT resource

When the user selects an AppleTalk connection from the Network Control Panel, the Network 'cdev' updates parameter RAM with a value that represents the AppleTalk connection that is currently selected. This value remains in parameter RAM after the user turns off the Macintosh.

At startup 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 connection 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.

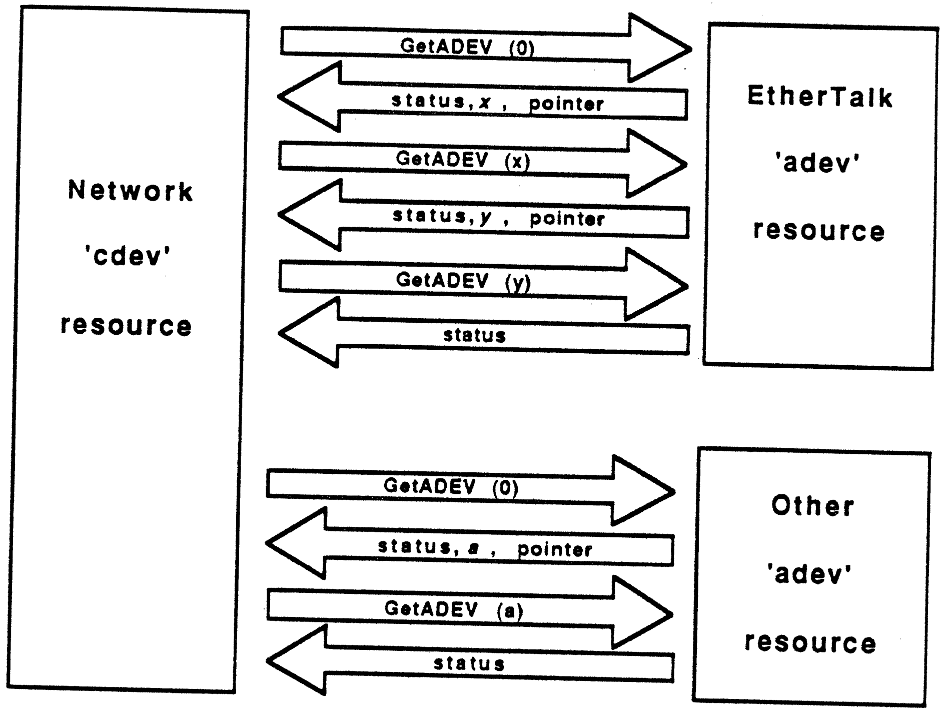
❖ *Note:* The LAP Manager INIT resource also loads the LAP Manager into memory and initializes the LAP Manager at startup time.

Calls to the 'adev' resource

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 icon in the Control Panel, the Network 'cdev' makes a series of GetADEV calls to each 'adev' resource in the System Folder. The 'adev' resource responds to each GetADEV with information about one connection that the 'adev' is supporting. The information includes a status and if appropriate, an 'adev'-dependent variable and a pointer to a string. The Network 'cdev' determines from the status information whether to send another GetADEV call to this particular 'adev' resource. If another GetADEV is sent to this 'adev', the variable returned from the previous GetADEV call retains its value; if not, the variable is reset to 0, indicating a first GetADEV call. The Network 'cdev' uses the pointer to determine the exact string to display under the icon for this particular connection. Figure 2-4 shows this interaction. The details of the GetADEV call are given in Chapter 3.



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

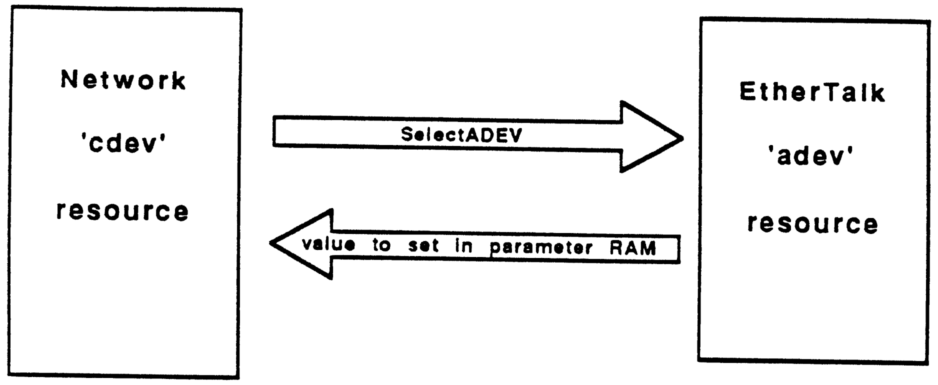
Figure 2-4
The GetADEV call

The SelectADEV call

When the user selects the icon for a particular AppleTalk connection from the Network Control Panel, the Network 'cdev' makes a SelectADEV call to the 'adev' resource to indicate the selection and to determine the value to place in parameter RAM. This value indicates the details of the selected AppleTalk connection.

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. The Network 'cdev' eventually passes this value to the 'atlk' resource and places it in parameter RAM, where it will be available to the LAP Manager INIT resource at startup time.

Figure 2-5 illustrates the SelectADEV call for EtherTalk.



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

Figure 2-5
The SelectADEV call

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

Calls to the 'atlk' resource

The Network 'cdev' also interacts with the 'atlk' resource when an AppleTalk connection is selected from the Network Control Panel.

After making the SelectADEV call to the 'adev' resource, the Network 'cdev' makes an AShutdown (AppleTalk Shutdown) call and then an AInstall (AppleTalk Install) call to the 'atlk' resource. The AShutdown call instructs the 'atlk' resource to close down a currently active AppleTalk connection. The AInstall call intstructs the 'atlk' resource to install a newly selected AppleTalk connection in the LAP Manager's LAPWrite hook.

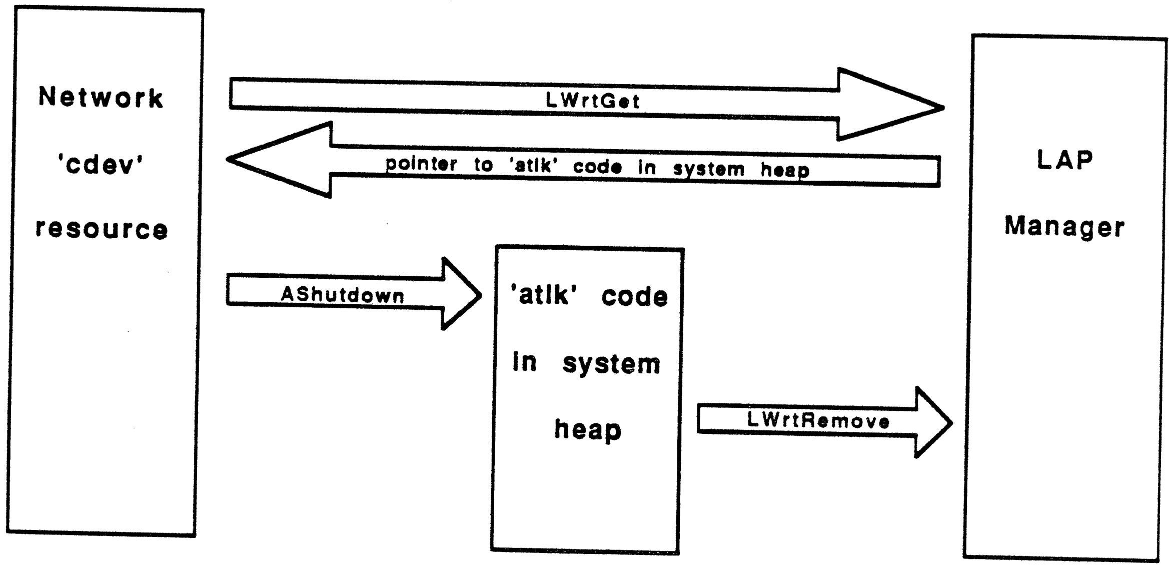
The 'atlk' resource responds to these calls by interacting with the LAP Manager to carry out the instructions.

❖ *Note:* The AInstall call and the AShutdown call each include a port number parameter. The LAP Manager can use this logical value to support several concurrently active connections. In the case of more than one active connection, the port number specifies the connection for which a call is intended. This will be useful in future bridging operations; however, the current version of the Network 'cdev' file does not support this functionality.

The LWrtGet, AShutdown, and LWrtRemove calls

To close down the previous AppleTalk connection, the Network 'cdev' makes an AShutdown call to dispose of the 'atlk' resource. However, by this time, the Network 'cdev' has detached the previously installed 'atlk' code and does not know the location of that code in the system heap. Before making the AShutdown call to the 'atlk' code, the Network 'cdev' makes a LWrtGet (LAP Write Get) 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 LWrtRemove (LAP Write Remove) call to the LAP Manager. The LAP Manager responds to this call by removing the old 'atlk' code from the LAPWrite hook.

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

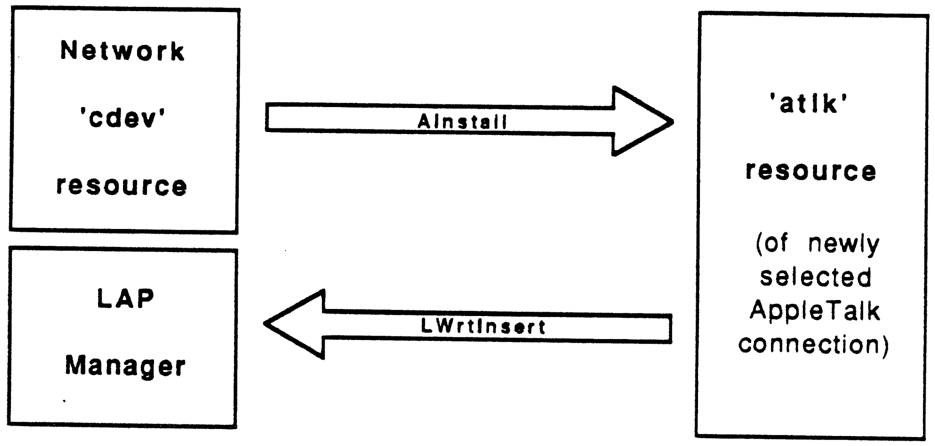
The AInstall and LWrtInsert calls

After the previous AppleTalk connection is shut down, 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 after the user closes the Control Panel.

Generally, in response to the AInstall call, the 'atlk' code makes an LWrtInsert (LAP Write Insert) call to instruct the LAP Manager to install a portion of the 'atlk' code into the *LAPWrite hook,* which is a low-memory location equal to ATalkHk2. This portion of 'atlk' code is responsible for sending packets.

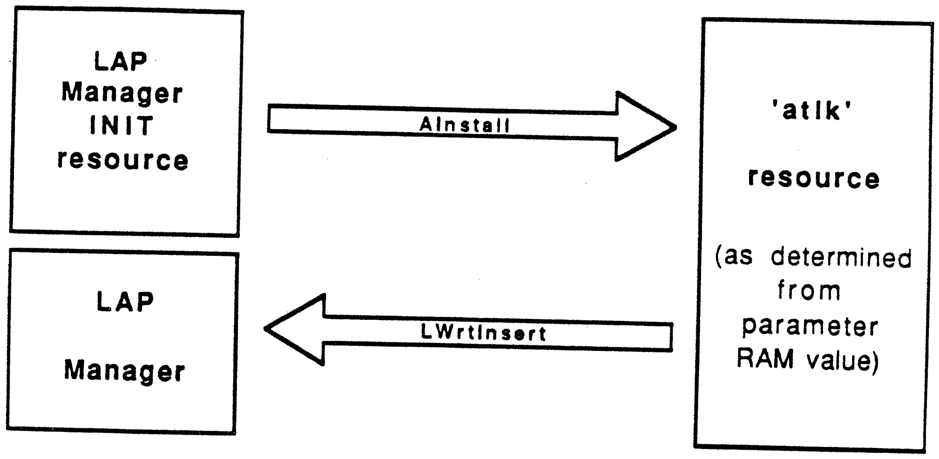
At startup time, the LAP Manager INIT resource also loads the 'atlk' resource, as indicated by parameter RAM, into the system heap; makes the AInstall call to the 'atlk' resource; and detaches the 'atlk' resource.

Figure 2-7 illustrates the AInstall and LWrtInsert calls when an AppleTalk connection is selected from the Control Panel. Figure 2-8 shows these same two calls at system startup time.



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

Figure 2-7
AInstall and LWrtInsert calls when an AppleTalk connection is selected


LAP Manager INIT resource
AInstall
'atlk' resource (as determined from parameter RAM value)
LAP Manager
LWrtInsert

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

Figure 2-8
AInstall and LWrtInsert calls at system startup time

The LAP Manager

The LAP Manager, which resides between the AppleTalk protocol stack and the Link Access Protocols of all AppleTalk connections, standardizes interactions between the protocol stack of the AppleTalk drivers and the LAP layer of the active AppleTalk connection. By standardizing these interactions, the LAP Manager ensures that various AppleTalk connections do not interfere with each other and do not access information that is private to the AppleTalk drivers.

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

AppleTalk connection

The LAPWrite hook contains the code that is, for all practical purposes, the actual AppleTalk connection for outgoing packets. In the case of EtherTalk, this code is installed in the LAPWrite hook when the 'atlk' resource makes the LWrtInsert call to the LAP Manager. The 'atlk' resource tells the LAP Manager which portion of the 'atlk' code to insert into the LAPWrite hook. Loading this code into the LAPWrite hook happens at two times:

- whenever the user selects an AppleTalk connection from the Network Control Panel
- at startup time when the LAP Manager's INIT resource obtains the value of the active AppleTalk connection from parameter RAM.

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 an LRdDispatch (LAP Read Dispatch) 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)

You can use AARP to resolve your network-addressing requirements, such as mapping between any two sets of addresses. A common application of AARP provides an AARP client, such as the Datagram Delivery Protocol (DDP), with a hardware address that corresponds to the protocol address of the node that is to receive the packet. AARP maintains a collection of these protocol-to-hardware address mappings. If AARP does not know the hardware address, AARP obtains the hardware address from the network. 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 reference refers to them as follows:

- An *Ethernet address*, which is the node address that is determined by the Physical and Link Access 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 what AARP generically refers to as the hardware address.
- An *AppleTalk address*, which is the node address used by higher-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 what AARP generically refers to as the 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 particular protocol stack.* This protocol address must be unique among all nodes on the network.
- *Mapping from a protocol address to a hardware address.* When given a protocol address for a node on the network, AARP returns either the corresponding hardware address or an error indicating that no node on the network has such a protocol address.
- *Filtering of packets.* For all data packets received by a 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. If the packet does not equal either of these values, AARP discards the packet.

Refer to Chapter 5 for a detailed explanation of AARP.

The Ethernet driver

While the Macintosh II uses the EtherTalk software, the Ethernet driver serves as a general-purpose interface between the 'atlk' resource and the EtherTalk Interface Card. You should use the driver interface with other Ethernet implementations, such as an interface to a Macintosh SE driver.

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 must obtain certain field values, such as the EtherTalk card slot number. You may obtain these field values by using the Slot Manager sNextsRsrc trap.

There are two transmission modes available for the Ethernet driver: **Limited-transmission mode** and **General mode**.

When the Ethernet driver first opens, the driver is in Limited-transmission mode. In Limited-transmission mode, the Ethernet 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 AppleTalk packets for transmission and to allow a larger buffer-pool area for packet reception. If packets to be sent are greater than 768 bytes, issue a control call (ESetGeneral) to change from Limited-transmission mode to General mode. In General mode, the driver can transmit any valid Ethernet packet. Both modes allow reception of any valid Ethernet packet.

Transmission and reception

A series of Device Manager control calls are made to the driver to control packet transmission and packet reception over Ethernet. The 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 a particular 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 the Limited-transmission mode 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 connection. The 'adev' file is similar to the 'cdev' file, and both reside in the System Folder.

When the user selects the Network 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 connections available to the user. In addition, the Network 'cdev' highlights the icon of the currently selected AppleTalk connection. If the user then selects an alternate AppleTalk connection, the Network 'cdev' highlights the icon of 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 'atlk' resource into the system heap, and initializes the 'atlk' code.

The 'adev' file contents

There is an 'adev' file for EtherTalk and for any other AppleTalk connection. Each 'adev'' is located in the System Folder, and must contain the resources and code segments shown in Figure 3-1.

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
Contents of the 'adev' file

In addition to the resources and code segments shown in Figure 3-1, 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

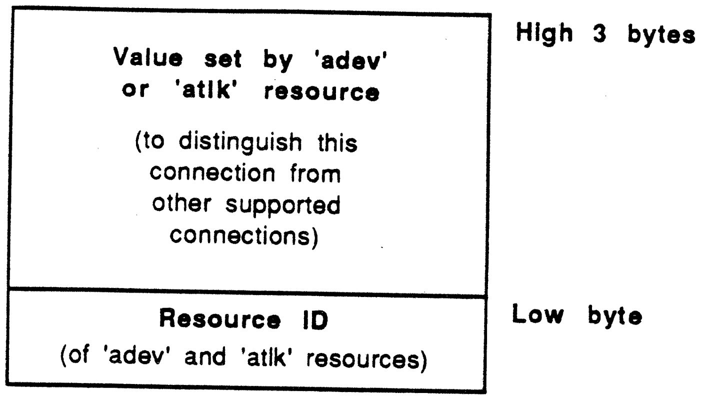
The 'adev' resource, located in an 'adev' file, is responsible for handling all interaction with the user. The Network 'cdev' loads the 'adev' resource into the application heap, calls the 'adev' resource, and removes it as needed.

The 'atlk' resource is responsible for the actual implementation of the alternate AppleTalk connection. When the user selects an AppleTalk connection from the Network Control Panel, the Network 'cdev' 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 must be in the range of 1 to 254. When stored in the low byte of parameter RAM, this ID identifies the currently selected AppleTalk connection.

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

Like drivers, no two AppleTalk connections 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.


Value set by 'adev' or 'atlk' resource
(to distinguish this connection from other supported connections)
High 3 bytes
Resource ID
(of 'adev' and 'atlk' resources)
Low byte

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

Figure 3-2
Parameter RAM value for AppleTalk connections

Calls to the 'adev' resource

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

- □ When the user selects the Network icon, the Network 'cdev' issues a series of GetADEV calls to construct the Network Control Panel.
- □ When the user selects an AppleTalk connection from the Network Contro Panel, the Network 'cdev' issues a SelectADEV call.

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 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 icon, the Network 'cdev' must display a list of icons to represent all alternate AppleTalk connections 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. The 'adev' resource responds to each GetADEV with information about one connection that the 'adev' is supporting. The information includes a status and if appropriate, an 'adev'-dependent variable and a pointer to a string. The Network 'cdev' determines from the status information whether to send another GetADEV call to this particular 'adev' resource. If another GetADEV is sent to this 'adev', the variable returned from the previous GetADEV call retains its value; if not, the variable is reset to 0, indicating a first GetADEV call. The Network 'cdev' uses the pointer to determine the exact string to display under the icon for this particular connection.

The Network 'cdev' displays the icon 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 by using a Slot Manager sNextsRsrc call and then can append the slot number to the string.

The first GetADEV call contains the current value of parameter RAM in D1 to indicate the currently selected alternate AppleTalk connection, 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 to indicate whether or not there are any cards to support. 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 the 'adev' resource supports at least one card, 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

The 'adev' resource may return one of three status-flag bytes in D0 to indicate the status of the alternate AppleTalk connections.

The 'adev' resource returns –1 in D0 to inform the Network 'cdev' that there is one and maybe more alternate AppleTalk connections (cards) supported by this 'adev' resource and that the current alternate AppleTalk connection *seems* to be selected, as indicated by parameter RAM. The Network 'cdev' responds by making another GetADEV call to the 'adev' resource.

Similarly, returning 0 in D0 informs the Network 'cdev' that there is one and maybe more alternate AppleTalk connections to support and that the current alternate AppleTalk connection is ***not*** selected, as indicated by parameter RAM. The Network 'cdev' responds by making another GetADEV call to the 'adev' resource.

Returning 1 in D0 informs the Network 'cdev' that there are no more alternate AppleTalk connections to support.

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

Before the 'adev' resource returns information about an alternate AppleTalk connection that 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 –1 in D0, the Network 'cdev' checks various system parameters and highlights the icon *only* if it is the current selection. The 'adev' resource may be wrong about identifying the current selection to the Network 'cdev'. For example, after two different AppleTalk connections examine the high 3 bytes of parameter RAM, both may return –1 in D0. To handle this possibility, the Network 'cdev' examines the low byte of parameter RAM, which contains the resource ID of the current selection, and matches the ID with the ID of the proper AppleTalk connection. 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 3 bytes)	Value to set in parameter RAM; also passed to the 'atlk' code by the AInstall call.

The Network 'cdev' makes a SelectADEV call to the associated 'adev' resource when the user selects the icon for an alternate AppleTalk connection. 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 connection, indicates the details of that connection and is passed to the 'atlk' resource by the AInstall call. For instance, in addition to the resource ID of the 'adev' resource and the 'atlk' resource, the high 3 bytes may contain the slot number of the interface card ($09–$0E). Depending on your application, you may 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 connection. 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.

After the AInstall call is made, if there is no error, the Network 'cdev' detaches the 'atlk' resource (from the Resource Manager) so it remains in the system heap after the user closes the Control Panel.

In the case of multiple concurrently active AppleTalk connections, a different copy of the 'atlk' code is loaded for each instance of active code.

The Network 'cdev' also makes an AShutdown call to dispose of the previously selected 'atlk' resource. Before making this call, the Network 'cdev' must 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' can use this location to determine where to call the 'atlk' code with AShutdown.

The AInstall and AShutdown calls use the D4 register to determine the port number. This extension is provided to permit multiple concurrently active AppleTalk connections. The value of D4 is generally 0 in a single-connection environment. In a multiple connection environment, its value is in the range of 0–*n*, where *n* equals the maximum number of connections minus 1. (As an example, for a Macintosh with six EtherTalk Interface Cards installed and concurrently active, there are six ports with port number values of 0–5).

For AInstall and AShutdown, 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 RTS.

The AInstall call (D0 = 1)

Call:	D1 (long)	Value from parameter RAM (as set in the SelectADEV call)
	D4 (byte)	Port number on which to perform the install operation (generally 0 in a single-connection environment)
Return:	D0	Error code
	D1 (high 3 bytes)	New value to set in parameter RAM

When either the Network 'cdev' or the 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 by using a LWrtInsert call to install itself as the alternate AppleTalk connection. The LAPWrtInsert call should return a value to set in parameter RAM only if that value is different than the one received; if the values are the same, 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; if an error does *not* occur, your code should return 0 in D0.

The AShutdown call (D0 = 2)

Call:	D4 (byte)	Port number on which to perform the shutdown operation (generally 0 in a single-connection environment)

The Network 'cdev' makes this call after the LAP Manager closes the AppleTalk drivers and before the Network 'cdev' installs a new alternate AppleTalk connection. The 'atlk' code should issue a LWrtRemove call to the LAP Manager, dispose of its variables, and perform any other necessary operations before the Network 'cdev' disposes of the 'atlk' resource. The 'atlk' code should return 0 in D0 to indicate no error.

Chapter 4

Calls to the LAP Manager

The LAP Manager standardizes interactions between the AppleTalk protocol stack and the Link Access layer of the currently selected AppleTalk connection. Standardizing these interactions ensures that various AppleTalk connections do *not* interfere with each other and do not need 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 connections and the remaining higher layers of the AppleTalk protocol stack.

The LAP Manager provides functions for installing the 'atlk' code in the LAPWrite hook, 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 .MPP driver contains code to implement various AppleTalk protocols, such as the Datagram Delivery Protocol and the Name Binding Protocol.

The LAP Manager also provides extensions to permit multiple concurrently active AppleTalk connections. The extensions include the addition of several calls for handling router operations and the port parameter, used in the AInstall and AShutdown calls and in certain other calls to the LAP Manger. The port parameter is a logical value that is used to uniquely identify a specific active connection when there are several active connections. The port is a numeric value in the range of zero to *n*, where *n* is equal to the number of allowed connections minus 1. The number of allowed connections is determined by the number of LAP interface cards (such as the EtherTalk Interface Card) that are installed in the Macintosh II.

This chapter describes the calls that the LAP Manager provides. After the Network 'cdev' or the LAP Manager INIT resource loads the 'atlk' resource into the system heap and makes the AInstall call, the 'atlk' code responds by making the LWrtInsert call to the LAP Manager, which then inserts the 'atlk' code into the LAPWrite hook. In the case of multiple AppleTalk connections, a different copy of the 'atlk' code is loaded for each instance of active code.

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

Calling the LAP Manager

The 'atlk' 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. The .MPP driver jumps through this location immediately before it writes a packet out through ALAP. If the user selects an alternate AppleTalk connection, the LAP Manager uses LAPMgrPtr to take control so it can call the alternate AppleTalk connection. The LAPMgrCall offset 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 the Macintosh Plus and later ROMs.

LAP Manager functions

The LAP Manager supports the nine functions described next 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 tries to get an unused node address (0 = infinite)
	D4 (byte)	Port on which to perform the operation (generally 0 in a single-connection environment)
Return:	D0	0 indicates no error

This call inserts an alternate AppleTalk connection into the LAPWrite hook. After the 'atlk' resource makes this call, the LAP Manager calls the code to which A0 points before the .MPP driver writes any packet to the network. Use the bits in 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.

❖ *Note:* All other bits should be set to 0.

The LAP Manager generally handles intranode-packet delivery (packets sent to your 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 sent to all nodes on the network (using the broadcast address), 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 B interrupts. Normally, the LAP Manager disables these interrupts because it assumes that the alternate AppleTalk connection does not want to receive ALAP packets on this port.

When you pick 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 connection does not distinguish between server addresses (128–254) and workstation addresses (1–127), and that the AppleTalk connection wants to pick a node address in the full range of addresses from 1 to 254.

The LAP Manager calls the code to which A0 points at two times. First, 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. Second, the LAP Manager calls the code 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 process is still in progress, and should jump to the location to which A0 points when the write process finishes. When the write process finishes, it must reset A1, A2, and D2 to their initial values, and must have preserved A4–A6 and D4–D7.

If the code wishes the .MPP driver to continue its normal processing in the situation where the 'atlk' code does *not* intercept the call, the 'atlk' code should jump to the location to which D1 points. Generally, the 'atlk' code does always intercept the call.

If D0 is nonzero, which indicates a call to send ENQs, the 'atlk' code should query if the address that D2 specifies is in use, and should return through A0 immediately. At any time thereafter, if the 'atlk' code discovers that the address is in use, the 'atlk' 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 a pointer is stored at location $2D8 (AbusVars) or that the port-use byte is at location $291 (PortBUse). Save these pointers for future use, as specified in the first ENQ call. This is especially important when multiple AppleTalk connections are concurrently active.

LWrtRemove (D0 = 3)

Return:	D0	0 indicates no error
	D4 (byte)	Port on which to perform the operation (generally 0 in a single-connection environment)

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

LWrtGet (D0 = 4)

Return:	D0	0 indicates no error
	D4 (byte)	Port on which to perform the operation (generally 0 in a single-connection environment)
	A0 —>	Start of code in the LAPWrite hook

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

LSetInUse (D0 = 5)

Call:	A2 —>	.MPP variables from the first ENQ call
Return:	D0	0 indicates no error

In a single active connection environment, the LSetInUse call indicates to both the LAP Manager and the .MPP driver that another node on the network is currently using the requested node address. The .MPP driver then tries another address. This call does not honor the port server/workstation bit (Bit 5) of the LWrtInsert call's Flag byte.

LNSetInUse (D0 = 12)

Call:	A2 —>	.MPP variables from the first ENQ call
	D4 (byte)	Port on which to perform the operation (generally 0 in a single-connection environment)
Return:	D0	0 indicates no error

In a multiple active connections environment, the LNSetInUse call indicates to both the LAP Manager and the .MPP driver that another node on the network is currently using the requested node address. The .MPP driver then tries another address. This call does honor the port server/workstation bit (Bit 5) of the LWrtInsert call's Flag byte.

LGetSelfSend (D0 = 6)

Call:	A2 —>	.MPP variables from the first ENQ call
Return:	D0	0 indicates no error
	D1 (byte)	Value of .MPP SelfSend flag

The LGetSelfSend call should be used by alternate AppleTalk connections that implement their own intranode delivery. If D1 is nonzero, intranode delivery is enabled.

LRdDispatch (D0 = 1)

Call:	A2 —>	.MPP variables

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

A0 —>	Hardware register (can be used by the alternate AppleTalk connections for any reason)
A1 —>	Hardware register (can be used by the alternate AppleTalk connections for any reason)
A2 —>	.MPP variables
A3 —>	Passed 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, does not return to the caller, but jumps to the protocol handler that is attached to the protocol indicated in D2. The protocol handler 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)	Global value of parameter RAM
	A0	Used internally

When more than one AppleTalk connection is concurrently active, there will be one global parameter RAM value and a local parameter RAM value for each active connection. The global parameter RAM value should be assigned at system configuration time, normally by the program that handles the routing functions. The connection represented by the global parameter RAM value is known as the *user's connection.* The local parameter RAM value is the current value that is resident in the LAP Manager's LAPWrite hook.

The LGetATalkInfo call returns the current global 4-byte value of parameter RAM.

The low byte of a parameter RAM value contains the resource ID of the 'adev' resource and the 'atlk' resource for the supported AppleTalk connection (0 for Built-in and 2 for Ethertalk). The high 3 bytes contain values that further distinguish this AppleTalk connection. (The 'adev' resource sets the high 3 byte values and passes them to the 'cdev' resource in response to the SelectADEV call, as described in Chapter 3 of this reference.)

LGetRouterInfo (D0 = 13)

Return:	D1 (word)	Volume reference number of the router volume (0, if no router is running)
	A1 —>	Pointer to the name of the router file, if any

This call is provided to support future software implementations in which multiple concurrently active AppleTalk connections are supported through a router.

❖ *Note:* This call is only available in LAP Manager version 2.0 and later.

LGetPortInfo (D0 = 10)

Call:	D4 (byte)	Port on which to perform the operation (generally 0 in a single-connection environment)
Return:	D1 (high 3 bytes)	Local value of parameter RAM, associated with the specified port.
	A0	Used internally

This call is provided to support future software implementations in which multiple concurrently active AppleTalk connections are supported through a router.

When more than one AppleTalk connection is concurrently active, there will be one global parameter RAM value and a local parameter RAM value for each active connection. The global parameter RAM value should be assigned at system configuration time, normally by the program that handles the routing functions. The connection represented by the global parameter RAM value is known as the ***user's connection.*** The local parameter RAM value is the current value that is resident in the LAP Manager's LAPWrite hook.

❖ *Note:* This call is only available in LAP Manager version 2.0 and later.

LShutdownRouter (D0 = 15)

Return:	D0	Non-zero value if error
	A0	Used internally

This call is provided to support future software implementations in which multiple concurrently active AppleTalk connections are supported through a router.

❖ *Note:* This call is only available in LAP Manager version 2.0 and later.

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	Nonzero if error
	A0	Used internally
	D2	Used internally

The LAARPAttach call is only used when you attach 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. Any application that wishes to receive AARP packets should use these two calls. Refer to Chapter 5 for more information about AARP packet-reception.

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
	D2	Used internally

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

Chapter 5

AARP and Data Packets

Depending on your application, you may decide to use AARP to resolve your network-addressing requirements. This chapter offers general information about the operation of AARP and specifically 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 stack that a node supports.. Normally, AARP is used in this way; however, you can also use AARP to maintain a collection of any two types of addresses.

Protocol stacks

A *protocol stack* is a collection of related protocols that correspond to the layers of the ISO-OSI reference model. Protocol stacks enable transmission and reception of packets over a network. AppleTalk protocols are an example of a protocol stack. Generally, information on the network is transferred between protocol stacks of the same type. For example, when a node transmits an AppleTalk packet, the packet is sent to a receiving node's AppleTalk protocol stack. Before a node sends a packet on the network, the node addresses the packet to the recipient by inserting a hardware destination address into the header section of the packet. Generally, a second address, called a *protocol address*, has already been inserted into the packet by one of the higher levels of the protocol stack. When used together, these two addresses identify both the node that is to receive the packet and the client within of the protocol stack for which the packet is intended.

Because a node may support more than one protocol stack, AARP maintains a collection of protocol-to-hardware address mappings for each protocol stack that a node supports. These address mappings are kept in an address mapping table (AMT), which AARP updates to ensure that current addressing information is available. The AMT serves as a cache of known protocol-to-hardware address mappings.

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 ALAP address. In addition to receiving packets addressed to a network's hardware address, a node may also receive packets that are addressed to the network's *broadcast-hardware* address or to a *multicast* address. If a sending node transmits a packet that contains a broadcast-hardware address as the destination address, all nodes on the network receive the packet. The network defines the broadcast-hardware address, which 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 receives the packet. Depending on the network configuration and the application, some nodes on the network may *not* have a multicast address, and other nodes may have one or more multicast addresses. Therefore, each node on the network receives all packets sent to the node's unique hardware address, to the broadcast-hardware address, and all packets sent to any multicast address group to which the node belongs.

Protocol addresses

A *protocol address* is the address that a protocol stack assigns to identify the protocol client that is to receive a packet. An example of a protocol address is the 8-bit AppleTalk protocol address that DDP and AARP use to verify that an incoming packet is intended for DDP. For EtherTalk, AARP randomly assigns an AppleTalk protocol address at initialization time and verifies that this protocol address is unique among all other AppleTalk 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 address, a protocol client (such as DDP) may also receive packets addressed to a *broadcast-protocol* address. Just as the broadcast-hardware address causes all nodes on the network to receive a packet at the physical level, the broadcast-protocol address causes all nodes on the network to receive a packet at the protocol-stack level. For example, addressing a packet with a broadcast-hardware address and a broadcast-protocol address for the AppleTalk protocol stack causes all nodes on the network to receive the packet. Only those nodes that support the AppleTalk protocol stack will process the packet. If a node supports more than one protocol stack, this node (or AARP) should assign a protocol address that corresponds to a protocol client for each stack.

Packet categories

Within any protocol stack, there are generally two categories of packets that a node may encounter on a network. This reference 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 functions). Data packets are those packets that contain information for processing by a protocol stack.

Obtaining an address

Generally, to send data packets on the network, a transmitting client requests from AARP the hardware address that corresponds to the protocol address of the node that is to receive the data packet. To provide its client with the desired hardware address, AARP attempts to retrieve this hardware address from the cache of addresses in the AMT. If the hardware address is in 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 generally resides between the Link Access layer and the Network layer of the network, and performs three basic functions for each protocol stack that AARP supports:

- *Initial determination of a unique protocol address for a particular protocol client.* This address must be unique among all nodes on the network.
- *Mapping from a protocol address to a hardware address.* When 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 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.

AARP operation

The following sections detail the operational concept of AARP. Each section contains two parts. The first part explains a general rule of AARP operation. The second part explains the way that EtherTalk implements this rule. In some cases, the wording of each explanation may be almost identical. For example, the only change of wording may be to change *hardware address* to *Ethernet address*. In other cases, the wording of each explanation differs greatly. Presenting information in this way allows you to refer to either explanation (or to both) to satisfy your address-resolution requirements.

The Address Mapping Table

Within a node, AARP maintains an Address Mapping Table (AMT) for each protocol stack that the 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.

For EtherTalk, AARP maintains an AMT for the AppleTalk protocol stack. This AMT 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, because AppleTalk addresses are 8 bits in length, the AMT contains no more than 256 address entries.

Choosing an address

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

For EtherTalk, AARP randomly picks an AppleTalk address at initialization time for the node that AARP supports. After checking the address of 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 protocol stack, and sets that address as the node's tentative protocol address. If a mapping for that address in the AMT already exists for the protocol stack, 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. After identifying an address that is not in the AMT, AARP verifies the uniqueness of the address as described in the following chapters.

For EtherTalk, 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 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

After AARP identifies a tentative protocol address for a protocol stack, AARP broadcasts a number of *probe packets* to determine whether or not any other node on the network is currently using the protocol address. A probe packet contains the tentative protocol address (for that protocol stack). Any node receiving a probe packet whose protocol address matches its protocol address must respond by sending an AARP response packet, as described in the next section of this reference.

For EtherTalk, after AARP identifies a tentative AppleTalk address, AARP broadcasts a number of probe packets that contain the tentative AppleTalk address to determine whether or not 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 stack that the node supports, AARP checks the protocol address that is associated with the protocol stack. If the tentative protocol address matches the receiving node's protocol address, the receiving node sends an AARP *response packet* to the probing node. (See the "Request Packets" section of this chapter for more information.) 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 makes the tentative protocol address permanent and returns this address to its client.

For EtherTalk, when a node receives an AARP probe packet, AARP 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 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 makes the tentative AppleTalk address 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 should select another random address. While sending a probe packet, a node should never respond to an AARP probe packet or to an AARP request packet.

Request packets

When an AARP client makes a request to determine the hardware address that corresponds to a protocol address for a protocol stack, 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. A request packet asks a node to supply the hardware mapping for the protocol address and the type of protocol stack that the request packet supplies.

For EtherTalk, 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 AARP needs an Ethernet mapping.

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 protocol stack. If the receiving node's protocol address for that protocol stack matches, the node responds by sending an AARP response packet to the requestor to indicate the protocol-to-hardware node-address-mapping information. 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, AARP retransmits the packet a specified 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.

For EtherTalk, 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 node responds by sending an AARP response packet to the requestor. The response packet contains the receiving node's Ethernet address. The requesting AARP maps this address in the AMT and returns the Ethernet address to AARP's client. If there is no reply within a specific time-out, AARP retransmits the packet a specified 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 stack that AARP supports. There are two reasons for this requirement. The first reason is that AARP must verify that an incoming packet is actually addressed to its client for that protocol stack. 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.

For EtherTalk, 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 to update the AMT. In addition, 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 stack to which the packet belongs determines the data packet's construction, the location of the destination address within a data packet varies for different protocol stacks. The AARP client must inform AARP of the location of the data packet's destination address for each protocol stack that AARP supports. The 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 any of the node's broadcast-protocol addresses, AARP must discard the packet and assume the originator sent this packet by mistake.

For EtherTalk 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 AppleTalk header matches the node's AppleTalk address or the broadcast-protocol value ($FF). If AARP determines that the destination address of the packet does not match the node's AppleTalk address or the broadcast-protocol address, AARP discards the packet.

Gleaning information

Incoming data packets generally contain the source hardware address and the source protocol address. AARP can learn the source hardware address and the protocol address from the packet, and can update the appropriate AMT. Obtaining mapping information in this way, a process known as *gleaning*, eliminates the need to send an additional request packet the next time that the node tries to communicate with the sending node.

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

AARP can also obtain source information 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 the source protocol address. AARP should always add this address information to its AMT, even if it does not answer this packet. AARP should *not* glean any source information from probe packets because this information is tentative.

For EtherTalk, incoming data packets contain the source Ethernet address and the source AppleTalk address. After AARP determines that the packet is intended for AARP's AppleTalk client, AARP obtains the source's AppleTalk-to-Ethernet address-mapping information and updates the AMT. Obtaining mapping information in this way eliminates the need to send an additional AARP request packet the next time that the node tries to communicate with the sending node.

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

Aging AMT entries

An AARP implementation may wish to age AMT entries. 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 hold an invalid hardware address in the AMT.

One method of aging AMT entries 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 the entry after removing it.

For EtherTalk, 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.

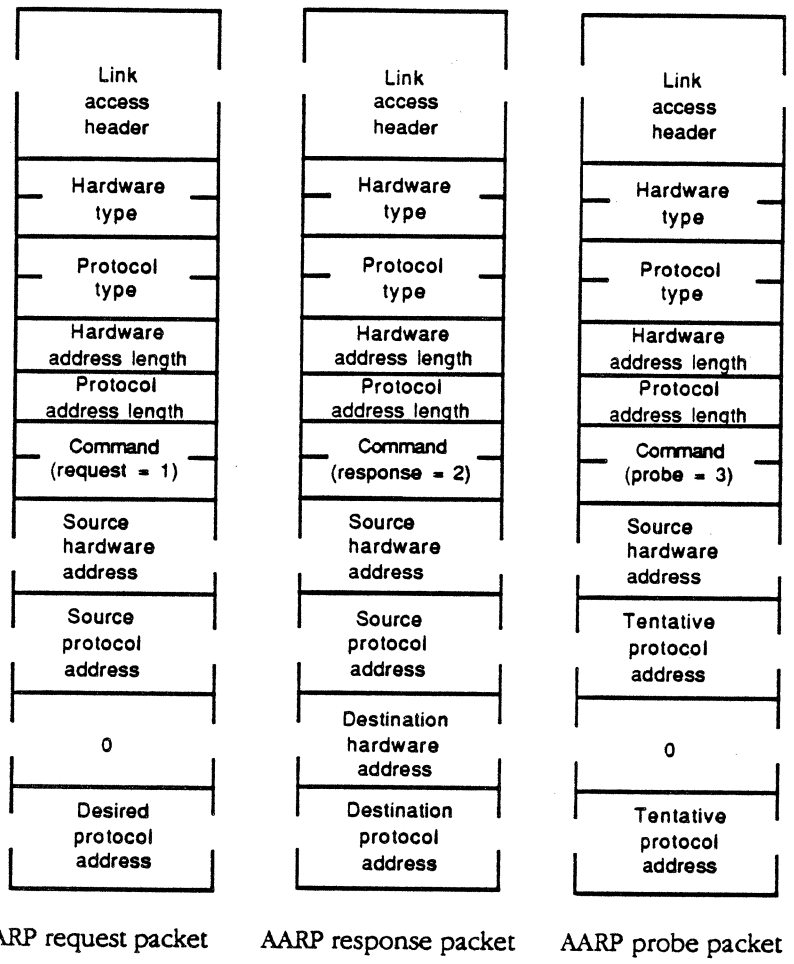
The age-on-probe process

Instead of using timed aging, you can 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. When timed-based aging is inefficient (when data packets do not reset the aging timer), AARP should implement this age-on-probe function for any node that does not glean address information from data packets.

For EtherTalk, AARP incorporates an age-on-probe function as well as timed aging. 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

Figure 5-1 presents the generic AARP packet formats.



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 particular medium in use (14 bytes for Ethernet). Following this header, there is header information that is a constant for the particular protocol-to-hardware mapping. This header information consists of the following:

- □ A 2-byte hardware type, which indicates the medium type (predefined)
- □ A 2-byte protocol type, which indicates the desired protocol stack (predefined)
- □ A 1-byte hardware-address length, which indicates the length in bytes of this field.
- □ A 1-byte protocol-address length, which indicates the length in bytes of this field.

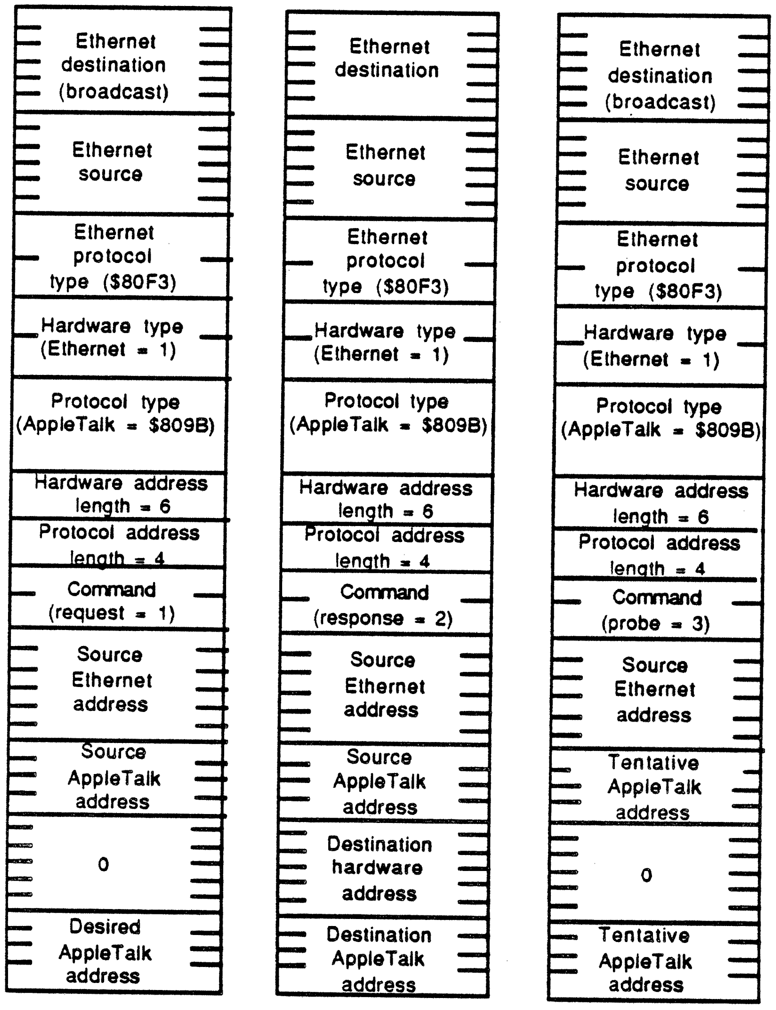
Following this header is a 2-byte command field that indicates the packet function (request, response, or probe). Next are the hardware and protocol addresses of the sending node (their lengths are specified in the length fields just described). Last in the packet are the hardware and protocol addresses of the receiving node.

In the case of an AARP request packet, the hardware address of the destination is unknown and should be set to a value of 0. 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 0.

AARP Ethernet-AppleTalk packet formats

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


Ethernet destination (broadcast)
Ethernet source
Ethernet protocol type ($80F3)
Hardware type (Ethernet = 1)
Protocol type (AppleTalk = $809B)
Hardware address length = 6
Protocol address length = 4
Command (request = 1)
Source Ethernet address
Source AppleTalk address
0
Desired AppleTalk address
Ethernet destination
Ethernet source
Ethernet protocol type ($80F3)
Hardware type (Ethernet = 1)
Protocol type (AppleTalk = $809B)
Hardware address length = 6
Protocol address length = 4
Command (response = 2)
Source Ethernet address
Source AppleTalk address
Destination hardware address
Destination AppleTalk address
Ethernet destination (broadcast)
Ethernet source
Ethernet protocol type ($80F3)
Hardware type (Ethernet = 1)
Protocol type (AppleTalk = $809B)
Hardware address length = 6
Protocol address length = 4
Command (probe = 3)
Source Ethernet address
Tentative AppleTalk address
0
Tentative AppleTalk address

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 5-2
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 the AARP packet:

- □ A 1-byte hardware type, which indicates Ethernet as the medium
- □ A 2-byte protocol type, which indicates the AppleTalk protocol
- □ A 1-byte hardware-address length, which indicates the length in bytes of the Ethernet address
- □ A 1-byte protocol-address length which indicates the length in bytes of the AppleTalk protocol 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 receiving node.

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 AARP needs an Ethernet address mapping.

For the probe packet, both the source AppleTalk address and the destination AppleTalk address should be set to the sending node's tentative AppleTalk address, and the destination hardware address should 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 retransmit count and interval depend on the desired thoroughness of the search. In general, AARP sets the probe-retransmission interval, but the request-packet-transmission interval can be assigned as a client-dependent parameter.

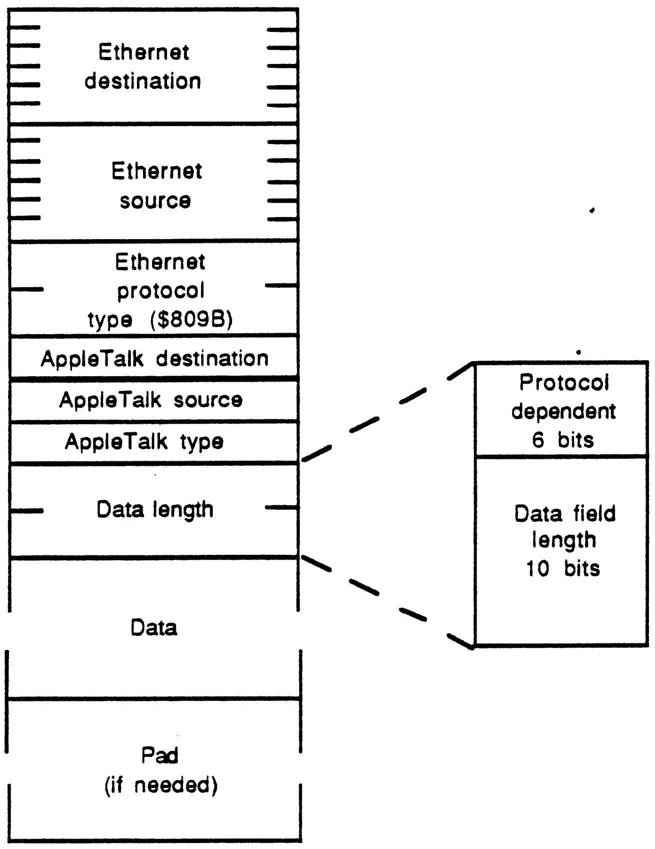
Packet specifics

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
Number of AARP probe retransmissions 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 contents of the high-order 6 bits are dependent on the higher-level protocol.

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 the bytes that make up the 14-byte Ethernet header, which is 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 future systems that may require such a header. All EtherTalk implementations must accept extended headers for any incoming AppleTalk packet (including packets that *Inside AppleTalk*, July 1986, identifies as requiring short headers). These implementations should also accept short DDP headers.

Chapter 6

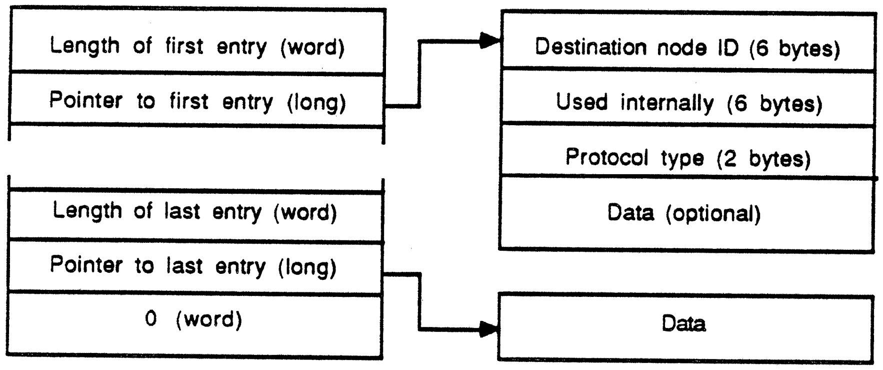
The Ethernet Driver

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 with the Macintosh II and the EtherTalk interface card; however, equivalent interfaces will probably be provided for other Ethernet interface cards and network drivers.

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 "The EWrite command" in this chapter for more information.) This command contains a pointer to a write-data structure (WDS). The WDS contains a series of length-pointer pairs that identify the lengths and memory locations of the packet's components. The WDS for Ethernet is shown in Figure 6-1.


Length of first entry (word)
Pointer to first entry (long)
Length of last entry (word)
Pointer to last entry (long)
0 (word)
Destination node ID (6 bytes)
Used internally (6 bytes)
Protocol type (2 bytes)
Data (optional)
Data

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 source address to replace the 6 unused bytes.

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

Protocol handlers

During a typical read operation, the interface card causes 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 the internal driver space and calling a piece of client code, known as a *protocol handler*, to process the rest of the packet. The 2-byte protocol type in the header tells the driver which protocol handler to call. The protocol handler responds by calling one or both of two driver routines (ReadPacket and ReadRest) to receive a packet.

The Ethernet driver also provides a general-purpose default protocol handler for use with the standard read call, which is described in "The ERead Command" in this chapter; however, you may decide to write you own protocol handler to receive a packet.

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 the packet piece-by-piece into a number of buffers, as long as the protocol handler 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 passing control to the protocol handler, the Ethernet driver passes various parameters and pointers in the processor's registers. Register setup and restrictions are essentially the same as those for ALAP protocol handlers (refer to *Inside Macintosh*, Volume II, for more information). The Ethernet driver calls the protocol handler as follows:

A0	Reserved for internal use by the driver (handler must preserve until ReadRest is complete)
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 2-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. An application can reduce D1 to indicate the number of pad bytes that are not read, but D1 should not otherwise be changed.

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	Equals 0 if requested number of bytes were read; is less than 0 or is greater than 0 if error
A0	Preserved
A1	Preserved
A2	Preserved
A3	Pointer to 1 byte past the last byte read

ReadPacket reads the number of bytes specified by D3 into the buffer to which A3 points. The remaining number of bytes to read is returned in D1. A3 points to the location where reading should begin 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), which can be 0

On exit

D0	Modified
D1	Modified
D2	Preserved
D3	Equals 0 if requested number of bytes were read; is less than 0 if packet was –D3 bytes too large to fit in buffer and was truncated; is greater than 0 if D3 bytes were not read (packet is smaller than buffer)
A0	Preserved
A1	Preserved
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
To avoid a system crash, always call ReadRest to read the last part of a packet.

If the protocol handler wishes to discard the remaining data before reading the last byte of the packet, the protocol handler 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 RTS, even if the driver returns an error. If the driver returns an error from a ReadPacket call, the protocol handler must quit via RTS without calling ReadRest at all. Upon return from ReadRest and ReadPacket, the zero (z) bit in the command control register is 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 must obtain certain field values, such as the card's slot number, before making this call. You can use the slot Manager to obtain these values.

Slot Manager sNextsRsrc trap macro

The Macintosh II has six slots, ranging from $09 to $0E. Built-in devices use slot 0. 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 0 for the sNextsRsrc fields spSlot, spID, and spExtDev, this routine returns the values for spId, spSlot, spCategory, and spType, in addition to other information for each card installed. The routine starts at slot $09 and continues to slot $0E. By matching the fields spCategory and spType 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 field spCategory as *CatNetwork* and the field spType as *TypEthernet.*

❖ *Note:* More information about the sResource Type formats is contained in the document *Macintosh Programmer's Workshop Reference*, version 2.0. Refer to *Inside Macintosh*, Volume V, for more information on the Slot Manager.

Refer to Appendix A for a code sample that shows how to use the sNextsRsrc trap.

Device Manager PBOpen call

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

—>	34	ioSlot	Byte
—>	35	ioId	Byte

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

When the driver opens, Ethernet packets for transmission can contain no more than 768 bytes, an amount that is more than sufficient to encapsulate an Ethernet-AppleTalk packet of the maximum size of 617 bytes. In Limited-transmission mode, the driver can allocate more space in the buffer pool for packet reception. If the packets that your application wishes to send require more than 768 bytes, you may change to General mode to transmit any valid Ethernet packet that is up to 1514 bytes long.

Here is an example of code that uses the sNextTypesRsrc trap to open the Ethernet driver:

```
            WITH      spBlock

            MOVEQ     #spBlockSize/2-1,D0          ;D0 = size of memory for
                                                   ;the Slot Manager
@10         CLR       -(SP)                        ;Clear word on stack
            DBRA      D0,@10                       ;Get whole block
            MOVE.L    SP,A0                        ;A0 -> slot block
            MOVE      #catNetwork,spCategory(A0)   ;Set category to network
            MOVE      #typEtherNet,spCType(A0)     ;Set category type to
                                                   ;Ethernet
            MOVE.B    #%0011,spTBMask(A0)          ;Mask off software and
                                                   ;hardware type
            _sNextTypesRsrc 0                      ;Get first one (slot and ID
                                                   ;are zero)
            MOVE      spSlot(A0),D1                ;Get slot, ID in low word
                                                   ;of D1

            ADD       #spBlockSize,SP              ;Deallocate slot block
            TST       D0                           ;Check for error
            BNE.S     @20                          ;Branch if none found
            LEA       OurQEl,A0                    ;A0 -> queue element
@15         MOVE      D1,ioSlot                    ;Set slot, ID in queue
                                                   ;element
            CLR.B     ioPermssn(A0)                ;Make sure permission is
                                                   ;clear
            LEA       ENETName,A1                  ;A1 -> name
            MOVE.L    A1,ioFileName(A0)            ;Set in queue element
            _Open     ,IMMED                       ;Open it (ioFlags zero)
@20         RTS                                    ;Return error code -
                                                   ;refnum in queue element
            ENDWITH

ENETName    DC.B      '.ENET'                      ;Name of Ethernet driver
                                                   ;for the Macintosh II
```

·Using an alternate Ethernet address

Instead of the hardware address that is built into the EtherTalk Interface Card, the Ethernet driver can use a software configurable address as the card's Ethernet address.

While opening for an EtherTalk Interface card, the Ethernet driver looks in the current resource chain for a resource of type 'eadr' that has a resource ID equal to the slot number (9, $A, $B, $C, $D or $E) of that card. A slotless Ethernet driver (.ENET0) should use resource ID 0 (zero) for its 'eadr' resource. The resource must not indicate a broadcast or multicast address.

If the 'eadr' resource is found, the Ethernet driver sets this 6-byte resource as the card's hardware address and uses it from then on.

Making commands to the Ethernet driver

After the Ethernet driver opens, the Device Manager makes a series of control calls to the driver to control packet transmission and reception. The calling code passes command arguments in the queue element that starts 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. If the packet size is less than 60 bytes (the Ethernet minimum), the driver adds pad bytes to the packet. The EWrite control call is made as follows:

Parameter block

—> 26	csCode	Word	{always EWrite}
—> 30	EWdsPointer	Pointer	{write-data structure}

Result codes

0	No error
elenErr	The packet was too large (packet is greater than 768 bytes when the driver is in Limited-transmission mode or packet is greater than 1514 bytes when the driver is in General mode) or the first WDS entry did not contain the full 14-byte header
excessCollsns	The packet could not be written because of a hardware error

The EAttachPH command

Use 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, the Ethernet driver uses a default protocol handler. The default protocol handler is for use with the ERead call, which is described in "The ERead Command" in this chapter). The EAttachPH command is made as follows:

Parameter block

—> 26	csCode	Word	{always EAttachPH}
—> 28	EprotType	Word	{Ethernet protocol type}
—> 30	Ehandler	Pointer	{protocol handler}

Result codes

0	No error
LAPProtErr	Protocol handler is already attached or table is full

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 0.

The EDetachPH command

Use the EDetachPH command to detach a protocol handler from the driver. This command is made as shown below:

Parameter block

—> 26	csCode	Word	{always EDetachPH}
—> 28	EprotType	Word	{Ethernet protocol type}

Result codes

0	No error
LAPProtErr	No protocol handler attached

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

The ERead command

Use the ERead call only to read 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 process, 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 removes the ERead call from the system queue, so more than one ERead call can be active concurrently. The ERead command is made as follows:

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}

Result codes

0	No error
LAPProtErr	Protocol not attached or protocol handler nonzero
buf2SmallErr	Buffer too small for whole packet; partial data returned
ReqAborted	Request terminated

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 *not* active, the ERdCancel call returns an error. If the ERead call is active, the ERead call completes with an error. The ERdCancel command is made as follows:

Parameter block

26	—>	csCode	{always ERdCancel}
28	—>	EKillQEl	{queue element pointer to cancel}

Result codes

0	No error
CBNotFound	ERead not active

The EGetInfo command

The EGetInfo command obtains driver information and takes arguments of a buffer pointer and size. In the first 6 bytes of the buffer, this call returns the Ethernet address for the node on which the driver is installed. The EGetInfo command is made as follows:

Parameter block

26	—>	csCode	{always EGetInfo}
28	—>	EBuffPtr	{buffer pointer}
34	—>	EBuffSize	{buffer size}

Result code

0	No error

For 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 receiving incoming packets
Bytes 11–14	Number of time-outs on transmitting outgoing packets
Bytes 15–18	Number of packets received that contain an incorrect address

The EAddMulti command

The EAddMulti command adds a multicast address to the list of accepted multicast addresses for this driver. The specified address must be a valid multicast address. After the EAddMulti command is issued, all packets that are directed to the specified multicast address are received in the same manner as packets that are directed to the node's hardware address.

Because more than one client of the Ethernet driver may use the same multicast address, the driver maintains a **use-count** for each of its multicast addresses. The use-count for a particular multicast address is incremented by 1 each time that an EAddMulti command specifying that multicast address is issued.

❖ *Note:* You should issue only one EAddMulti command for each multicast address that you wish to use.

The EAddMulti command is made as follows:

Parameter block

26	—>	csCode	{always EAddMulti}
28	—>	EMultiAddr	{Multicast address}

Result code

0	No error
eMultiErr	Invalid address or table full

The EDelMulti command

The EDelMulti command decrements the use-count for the specified multicast address and possibly removes the address from the list of accepted multicast addresses. The specified address must be a valid in-use multicast address for this driver.

Because more than one client of the Ethernet driver may use the same multicast address, the driver maintains a use-count for each of its multicast addresses. The use-count for a particular multicast address is decremented by 1 each time that an EDelMulti command specifying that multicast address is issued. When the use-count for a particular multicast address equals 0, that multicast address is removed from the list of accepted multicast addresses for this driver and packets that are directed to that multicast address are not received by this node.

❖ *Note:* Because there may be other clients that are using a particular mulicast address, you should issue only one EDelMulti command for each multicast address that you no longer wish to use.

The EDelMulti command is made as follows:

Parameter block

26	—>	csCode	{always EDelMulti}
28	—>	EMultiAddr	{Multicast address}

Result code

0	No error
eMultiErr	Address not found

The ESetGeneral command

The ESetGeneral command changes the driver from Limited-transmission mode to General mode. The command has no arguments. There is no command to change the driver from General mode to Limited-transmission mode. Changing the driver's mode may require a hardware reset and could cause loss of an incoming packet. The ESetGeneral command is made as follows:

Parameter block

26	—>	csCode	{always ESetGeneral}

Result code

0	No error

Chapter 7

The EtherTalk Interface Card

This chapter presents an overview of the operation of the EtherTalk Interface Card and explains each major component on the card. In addition, this chapter identifies the address assignments for local memory and gives the address assignments for the network interface controller (NIC) register.

❖ *Note:* Because of the design of the Ethernet drivers for the Macintosh operating system and the A/UX operating system, in these environments, you control the operation of the EtherTalk Interface Card without needing any knowledge of card operation. The information in this chapter becomes useful if you are developing drivers for environments other than the Macintosh operating system or the A/UX operating system.

About the EtherTalk Interface Card

The EtherTalk Interface Card installs in any slot for the NuBus™ interface on a Macintosh II computer. The card provides the interface between the Ethernet driver and the Ethernet cable system to enable packet transmission and reception among Ethernet nodes. The EtherTalk Interface Card may also function in the A/UX environment, transporting packet information using the Transmission Control Protocol/Internet Protocol (TCP/IP).

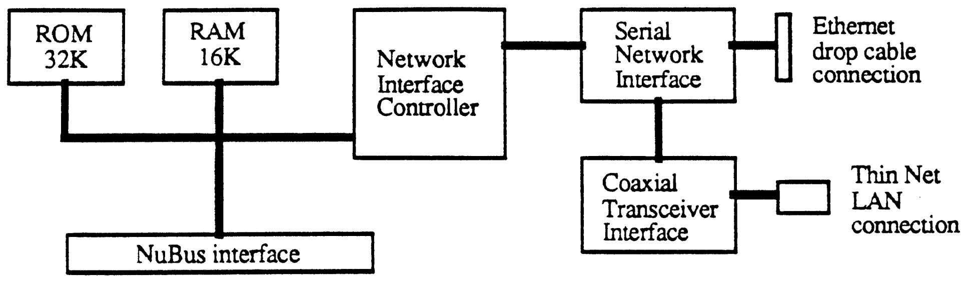
❖ *Note:* The A/UX operating system currently does *not* support EtherTalk software.

A detailed discussion of the A/UX operating system or device drivers is beyond the scope of this document. 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.

EtherTalk Interface Card hardware description

The EtherTalk Interface Card is a non-intelligent Ethernet adapter for the Macintosh II. The card uses a local-area-network (LAN) integrated-circuit set from National Semiconductor Corporation. The three LAN integrated circuits, or chips, are the Network Interface Controller (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.


ROM 32K
RAM 16K
Network Interface Controller
Serial Network Interface
Ethernet drop cable connection
Coaxial Transceiver Interface
Thin Net LAN connection
NuBus interface

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

Figure 7-1
EtherTalk Interface Card architecture

The EtherTalk Interface Card uses Apple's implementation of the NuBus™ interface. You can find more information on Apple's implementation of NuBus in the Apple publication *Macintosh II and Macintosh SE Cards and Drivers*, available from the Apple Programmers and Developers Association (APDA™).

The CTI chip is used as a coaxial line driver and receiver for Thin Net LANs. You do not use the CTI chip when attaching to an Ethernet backbone cable by means of an external transceiver interface. To choose between an Ethernet backbone cable connection or Thin Net LAN connection, make a jumper selection on the EtherTalk card. (Refer to the *Ethertalk Interface Card* document for information on making jumper selections.)

The SNI chip provides the bit-level encoding and decoding functions for Ethernet or Thin Net LANs. The SNI also provides a collision-signal translator and a diagnostic-loopback circuit.

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

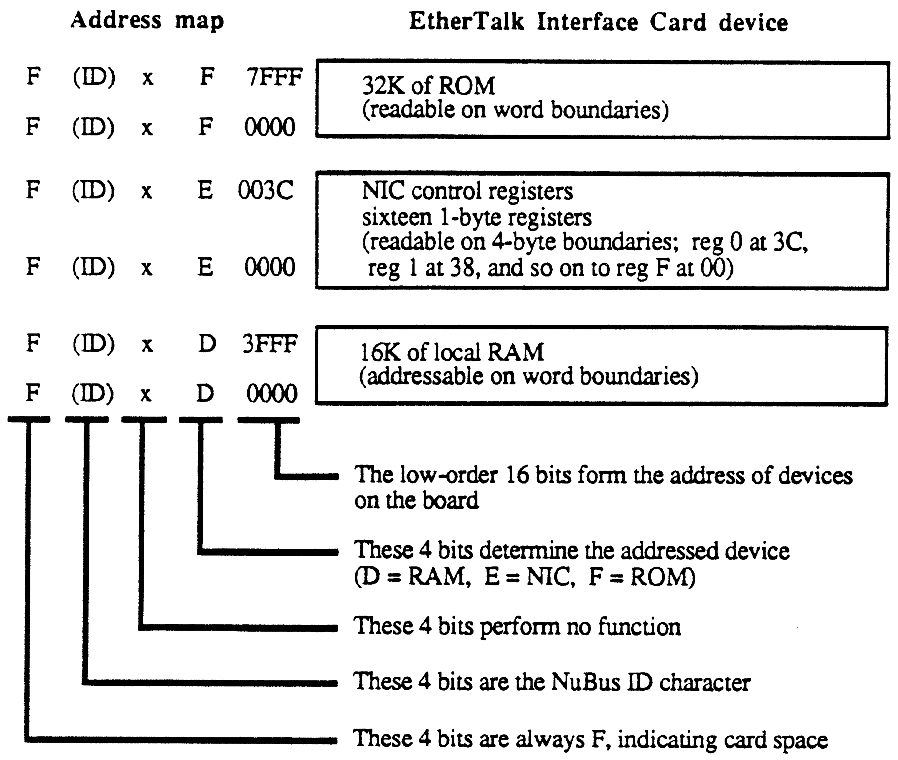
Local memory

The local memory consists of 16K of static RAM that is divided into a *transmit buffer* and a *receive buffer* by setting registers in the NIC. The buffers are further divided into 256-byte *pages*. The driver allocates the number of pages that make up the transmit buffer. The remaining pages are used for the receive buffer, which is a circular buffer (or *ring buffer*) established by Page Start and Page Stop registers. As the last address (set up by the Page Stop register) is reached, the next memory location used is the start of the buffer, as set up by the Page Start register, forming a continuous address space.

The local ROM memory is 32K in size and contains the Ethernet address and the NuBus slot information. Direct access to the ROM is not usually necessary because the Slot Manager provides these services for the Macintosh O/S and the slot library provides them for the A/UX operating system.

Address assignments

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


Address map
EtherTalk Interface Card device
F (ID) x F 7FFF
F (ID) x F 0000
32K of ROM (readable on word boundaries)
F (ID) x E 003C
F (ID) x E 0000
NIC control registers sixteen 1-byte registers (readable on 4-byte boundaries; reg 0 at 3C, reg 1 at 38, and so on to reg F at 00)
F (ID) x D 3FFF
F (ID) x D 0000
16K of local RAM (addressable on word boundaries)
The low-order 16 bits form the address of devices on the board
These 4 bits determine the addressed device (D = RAM, E = NIC, F = ROM)
These 4 bits perform no function
These 4 bits are the NuBus ID character
These 4 bits are always F, indicating card space

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. Use the NIC Command register to select register 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 a 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 7-1 displays the NIC Page 0 register addresses on the EtherTalk Interface 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

In Page 0, 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 7-2 displays the NIC Page 1 register addresses on the EtherTalk Interface 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 pointer (CURR)	Current pointer (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

Generally, the operating system reads the EtherTalk Interface Card ROM and installs 6 bytes into the Physical Address registers 0 through 5. The Current pointer is used to detect packet reception and to identify a page where a packet is currently being received. Use the Multicast Address registers to identify the multicast-address groups for the card.

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 (ALRT) and associated dialog item lists (DITL) are used at startup time to indicate an error occurred while installing the AppleTalk connection; 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 and other code to install the alternate AppleTalk connection at startup time
System Folder			Network	Network 'cdev' file; contains code to support a Control Panel choice of network connections
			EtherTalk	EtherTalk 'adev' file; contains code to implement EtherTalk when selected from the Control Panel

Ethernet driver equates

Here are the equates for the .ENET driver:

■ 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
EAddMulti       EQU 246            ;Add a multicast address
EDelMulti       EQU 245            ;Delete a multicast address
```

■ ENET queue element standard structure, in which arguments are passed in CSParam

```
EProtType       EQU CSParam        ;Offset to protocol type code
EMultiAddr      EQU CSParam        ;Offset to multicast address
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
```

■ Errors

```
ELenErr         EQU -92            ;Length error
EMultiErr       EQU -91            ;Multicast error
```

LAP Manager equates

Here are the equates for the LAP Manager:

■ **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       EQU 2          ;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       ;Points to start
LAPMgrCall     EQU 2          ;Offset to make LAP Manager calls
ATalkPRAM      EQU $E0        ;Start of parameter RAM
LAPMgrByte     EQU $60        ;Value of byte pointed to by LAPMgrPtr
```

■ **Resource ID**

```
adevBaseID     EQU -4032      ;Base resource ID for ADEVs
```

AARP equates

Here are the equates for AARP:

■ 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-specific equates

```
H_Ethernet    EQU 1                 ;Hardware type for Ethernet
HL_Ethernet   EQU 6                 ;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+       ;Source protocol address
              HL_Ethernet
AAEDstPhys    EQU AAESrcLog+        ;Destination hardware address
              PL_AppleTalk
AAEDstLog     EQU AAEDstPhys+       ;Destination protocol address
              HL_Ethernet
AAEEnd        EQU AAEDstLog+        ;End of packet
              PL_AppleTalk
```

■ 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

This section contains four code samples that make up an AppleTalk device file called *SampleNet.* This code executes on the Macintosh O/S. This is only a shell; you will have to develop the remainder of the 'atlk' resource.

'adev' resource

Here is a sample 'adev' resource:

```
*========================================================================
*   SNadev.a
*
*   Sample adev resource for implementing an alternate AppleTalk connection
*
*   Copyright © 1987 Apple Computer, Inc.   All rights reserved.
*========================================================================

                    PRINT       OFF
                    INCLUDE     'lapmgrequ.a'
                    INCLUDE     'Traps.a'
                    INCLUDE     'SlotEqu.a'
                    PRINT       ON

*------------------------------------------------------------------------
*   This is the entry point for our adev resource.  We're called here with
*   a selector in D0 which determines what type of call we've received.
*------------------------------------------------------------------------
adevStart   MAIN
            WITH        SpBlock
            CMPI.L      #GetADEV,D0         ; is it a GetADEV call?
            BEQ.S       doGetADEV           ; yes, go do it
            CMPI.L      #SelectADEV,D0      ; is it a SelectADEV call?
            BEQ.S       doSelectADEV        ; yes, go do it
            RTS                             ; received a message we don't
                                            ; know about, return
```

```
*------------------------------------------------------------------------
*  GetADEV
*    At this point we know we've received a GetADEV call.  We need to tell
*    the Network cdev which icons to display for our alternate AppleTalk
*    connection (if any).  For this sample alternate AppleTalk connection,
*    we're going to pretend that we can run over any card installed in the
*    machine (so that no matterwhat cards you have in your MacII, this
*    sample adev will do something)
*
*    To do this, each time we receive a GetADEV call, we call _Snextsrsrc to
*    determine our NuBus configuration.  (For more info on _Snextsrsrc,
*    see Inside Mac volume 5 - The Slot Manager).  In most cases, an
*    adev will call _Snexttypesrsrc instead of _Snextsrsrc.  _Snexttypesrsrc
*    allows you to find a particular category and type of board.  For example
*    you could look for boards with category=Network and type=Ethernet.
*------------------------------------------------------------------------
doGetADEV
             LINK         A6,#-spBlockSize     ; allocate an spBlock for the
                                               ;_Snextsrsrc call
             LEA          -spBlockSize(A6),A0  ; point A0 at our spBlock
             ADDQ         #1,D2                ; we want to start at the next
                                               ; card
             MOVE.B       D2,spSlot(A0)        ; fill in the slot number
             _Snextsrsrc
             TST.W        D0                   ; did we get an error?
             BNE.S        noMas                ; if we did, there are no more
                                               ; cards - return
             MOVE.B       spSlot(A0),D2        ; next value for D2 to call -
                                               ; it's the slot number

             LEA          iconString,A0        ; load up our string to display
             ADD.B        #$28,D2              ; If we add $28 to the slot
                                               ; number $09-$0E
             MOVE.B       D2,9(A0)             ; we end up with the decimal
                                               ; numbers 1-6
             SUB.B        #$28,D2              ; which we insert in the string
                                               ; we display

             ASR.L        #8,D1                ; shift the current value of PRAM
                                               ; one byte right
             CMP.B        D2,D1                ; are we the currently selected
                                               ; AppleTalk connection?
             BNE.S        @10                  ; no, just exit with D0=0 (we
                                               ; know from TST.W above)
             MOVE.B       #-1,D0               ; yes, tell the Network cdev we
                                               ; are

    @10:     UNLK         A6                   ; deallocate our spBlock
             RTS                               ; all done with GetADEV

noMas        MOVE.B       #1,D0                ; tell the cdev we have no more
                                               ; cards
             UNLK         A6                   ; deallocate our spBlock
             RTS                               ; all done with GetADEV
```

```
*-------------------------------------------------------------------------
*  SelectADEV
*    At this point we know we've received a SelectADEV call.  We get, in D2,
*    the value of the selected icon from the associated GetADEV call.  In
*    this case it's the slot number of the card selected.  We need to return
*    in the high three bytes of D1, a value we'd like stored in parameter
*    RAM.  All we do is shift the slot number up one byte and then return
*    it in D1.
*-------------------------------------------------------------------------
doSelectADEV
             ASL.L         #8,D2                  ; shift the slot number left one
                                                  ; byte
             MOVE.L        D2,D1                  ; put this value in D1 and return
             RTS                                  ; all done with SelectADEV
*-------------------------------------------------------------------------
*   This is the string we display under each icon.
*-------------------------------------------------------------------------
             STRING        PASCAL
iconString   DC.B          'SampleNet(X)'         ; hard coded string for our
                                                  ; sample adev
             ENDMAIN
             END
```

'atlk' resource

Here is a sample 'atlk' resource:

```
*=========================================================================
*   SNatlk.a
*
*   Sample atlk resource for implementing an alternate AppleTalk connection
*
*   Copyright © 1987 Apple Computer, Inc.   All rights reserved.
*=========================================================================

             PRINT     OFF
             INCLUDE   'lapmgrequ.a'
             PRINT     ON

*-------------------------------------------------------------------------
*   This is the main entry point for the atlk resource.
*   An 'atlk' resource 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 connection.
*   In this example, the first section of code branches to LAPStart.
*
*   The second section, located at the start of the resource plus two,
*   contains the AInstall and AShutdown routines.
*-------------------------------------------------------------------------
atlkStart    MAIN
             BRA.S         LAPStart             ;
             CMPI.L        #AInstall,D0         ; is it an AInstall call?
```

```
            BEQ.S           doAInstall              ; yes, go do it
            CMPI.L          #AShutdown,D0           ; is it an AShutdown call?
            BEQ.S           doAShutdown             ; yes, go do it
            MOVEQ           #-1,D0                  ; indicate an error
            RTS                                     ; received a message we don't
                                                    ; know about, return
*-----------------------------------------------------------------------
* AInstall
*   We're being installed as the alternate AppleTalk connection.  We're passed,
*   in D1, the value from parameter RAM (as set in the SelectADEV call).
*   In our case this is the slot number of the board selected.  Typically
*   at this point you'll allocate any variables you need or open the
*   appropriate I/O device (such as the slot driver).  In addition to the
*   above, we also need to call the LAP manager with a LWrtInsert call
*   here to install ourselves as the alternate AppleTalk connection.  Note that
*   we have to preserve D1 in this call.
*-----------------------------------------------------------------------
doAInstall
            MOVE.L          D1,-(SP)                ; save D1 on the stack
            LEA             atlkStart,A0            ; set up A0 to point at the
                                                    ; beginning of our 'atlk'
            MOVE.B          #0,D1                   ; clear all the flags
            MOVE.W          #8,D2                   ; try 8 times to get an unused
                                                    ; node id
            MOVE.W          #LWrtInsert,D0          ; we want to make a LWrtInsert
                                                    ; call
            MOVE.L          LAPMgrPtr,A1            ; A1-> start
            JSR             LAPMgrCall(A1)          ; entry point for LAP Manager
                                                    ; calls (returns D0)
            MOVE.L          (SP)+,D1                ; restore D1 from the stack
            RTS                                     ; done with AInstall
*-----------------------------------------------------------------------
* AShutdown
*   We're being removed as the alternate AppleTalk connection.  At this point
*   we need to dispose of any variables we've allocated, and then make a
*   LWrtRemove call to remove ourselves as the alternate AppleTalk connection.
*-----------------------------------------------------------------------
doAShutdown
            MOVE.W          #LWrtRemove,D0          ; we want to make a LWrtInsert
                                                    ; call
            MOVE.L          LAPMgrPtr,A1            ; A1-> start
            JSR             LAPMgrCall(A1)          ; entry point for LAP Manager
                                                    ; calls (returns D0)
            RTS                                     ; done with AShutdown
```

```
*-----------------------------------------------------------------------
*   This is the actual LAP code.  It is called at two different times.
*   The first is at address choosing time.  The LAP Manger calls this
*   code for each set of ENQs that the 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 the ALAP.
*-----------------------------------------------------------------------
LAPStart
                    RTS                            ; done with LAPStart
                    ENDMAIN
                    END
```

Makefile code

Here is the sample makefile code:

```
#===================================================================
#       Makefile for SampleNet
#
#       Copyright © 1987 Apple Computer, Inc.  All rights reserved.
#===================================================================

AOptions = -case on

SampleNet     ƒƒ     SampleNet.r SNadev SNatlk
       Rez SampleNet.r -o SampleNet
       Setfile -a B -t 'adev' -c 'Sample' SampleNet

# Note that we set the resLocked bit of the adev resource
SNadev ƒƒ     SNadev.a.o
       Link                    ∂
              -rt adev=126 ∂
              -ra Main=$10 ∂
              -t     rsrc   ∂
              SNadev.a.o    ∂
              -o SNadev

# Note that we set the resLocked and resSysHeap bits of the atlk resource
SNatlk ƒƒ     SNatlk.a.o
       Link                    ∂
              -rt atlk=126 ∂
              -ra Main=$50 ∂
              -t     rsrc   ∂
              SNatlk.a.o    ∂
              -o SNatlk
```

Resource file

Here is the resource file example:

```
/*  SampleNet.r

        Sample resource file for implementing an alternate AppleTalk connection

        Copyright © 1987 Apple Computer, Inc.  All rights reserved.
*/

#include "Types.r"

include "SNadev" 'adev' (126);    /* This is the alternate AppleTalk ID */
include "SNatlk" 'atlk' (126);    /* assigned by Macintosh Developer Technical
                                     Support */


resource 'BNDL' (-4032) {
        'Sample',
        0,
        {       /* array TypeArray: 2 elements */
                /* [1] */
                'ICN#',
                {       /* array IDArray: 1 elements */
                        /* [1] */
                        0, -4032
                },
                /* [2] */
                'FREF',
                {       /* array IDArray: 1 elements */
                        /* [1] */
                        0, -4032
                }
        }
};

resource 'FREF' (-4032) {
        'adev',
        0,
        ""
};
```

```
resource 'ICN#' (-4032) {
	{	/* array: 2 elements */
		/* [1] */
		$"FFFF FFFF 8000 0001 8000 0001 A1F0 0001"
		$"B208 0001 AC24 0001 AC0C 0001 B230 0001"
		$"A1F0 1001 8000 2801 8000 4801 803F 8C19"
		$"8040 9229 8090 A149 8100 0091 8130 0091"
		$"80C8 0149 8010 0229 803F FC19 8000 0001"
		$"8000 0001 8000 0001 8000 0001 8004 3FC1"
		$"8006 4021 8005 8091 8005 8011 8006 40E1"
		$"8004 3F01 8000 00C1 8000 0001 FFFF FFFF",
		/* [2] */
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
		$"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
	}
};

resource 'STR ' (-4032) {
	"SampleNet"
};

type 'Sample' {
	pstring;
};
resource 'Sample' (0) {
	"Example Alternate AppleTalk adev file.  ©1987, Apple Computer, Inc."
};
```