This chapter provides an overview of the AppleTalk networking system and the AppleTalk Manager. AppleTalk is a communications network system interconnecting personal computer workstations, computers acting as file servers and print servers, printers, and shared modems allowing them to exchange information through a variety of types of communications hardware and software. The AppleTalk Manager consists of a set of programming interfaces to the various components of AppleTalk for applications and processes running on Macintosh computers.

This chapter introduces some of the AppleTalk terminology that is used throughout the rest of this book. Read this chapter if you want to gain an overview of the AppleTalk networking system and its component protocols. You should also read this chapter for suggestions on which AppleTalk protocols to use for various application requirements.

- This first section of this chapter, "About Networking on the Macintosh," provides an introduction to AppleTalk networking concepts and terminology, and then it discusses
  - the AppleTalk protocols and their functions
  - the AppleTalk Manager
  - the layers of the Open Systems Interconnection (OSI) model and how the AppleTalk protocol stack relates to this model
- The second section of this chapter, "Deciding Which Protocol to Use," discusses how you can use each of the AppleTalk protocols that has an application programming interface.
- The third section of this chapter, "The AppleTalk Pascal Interface," describes the two modes in which you can execute the routines that make up the interfaces to the AppleTalk protocols. This information applies to each of the protocols covered individually throughout the chapters of this book. You should read this section before you use any of the programming interfaces to the AppleTalk protocols.

The chapters that make up the rest of this book describe how to use the AppleTalk Manager and the hardware device drivers. Because the AppleTalk network system includes both hardware and software—and because the software includes not only the AppleTalk Manager but also file servers, print servers, internet routers, drivers for circuit cards, and so forth—the information in this book constitutes only a small part of the body of literature documenting AppleTalk.

About Networking on the Macintosh

Networking on the Macintosh is implemented through AppleTalk. Applications and processes can communicate across a single AppleTalk network or an AppleTalk internet, which is a number of interconnected AppleTalk networks. Using AppleTalk, applications and processes can transfer and exchange data and share resources.

The AppleTalk networking system includes a number of protocols arranged in layers, which are collectively referred to as the ***AppleTalk protocol stack.*** Each of these protocols provides a set of functions and services that a protocol above it can use and build upon. A higher-level protocol is considered a ***client*** of the protocol that is below it in the AppleTalk protocol stack. (For information on how these protocols are implemented, see "The AppleTalk Protocol Stack" beginning on page 1-11.)

Many of the AppleTalk protocols provide application programming interfaces that you can use to access the services of the protocol. The programming interfaces to these protocols are collectively referred to as the ***AppleTalk Manager.***

This section provides

- an introduction to some AppleTalk networking fundamentals, including a discussion of addressing in AppleTalk
- an overview of the AppleTalk protocol stack, with a brief discussion of each protocol
- an overview of the AppleTalk Manager, which includes the LAP Manager programming interface

AppleTalk Networking

This section introduces some networking concepts and terms that pertain to AppleTalk and that are used throughout the chapters of this book. It discusses

- fundamental networking concepts and AppleTalk
- addressing in AppleTalk
- AppleTalk connectivity

Basic AppleTalk Networking Concepts

A networking system, such as AppleTalk, consists of hardware and software. Hardware on an AppleTalk network includes physical devices such as Macintosh personal computer workstations, printers, and Macintosh computers acting as file servers, print servers, and routers; these devices are all referred to as ***nodes*** on the network.

AppleTalk interconnects these nodes through transmission paths that include both software and hardware components. The software that governs the transfer of data across a computer network is commonly designed using a layered architecture or model. (For more information on networking models and AppleTalk, see "AppleTalk and the OSI Model" beginning on page 1-19.)

For each layer of a model, protocols exist that specify how the networking software is to implement the functions which that layer provides and interact with the layer above and below it. A ***protocol*** is a formalized set of procedural rules for the exchange of information and the interactions between the network's interconnected nodes. A network software developer implements these rules in programs that carry out the functions specified by the protocol. AppleTalk consists of a number of protocols, many of which are implemented in software programs called *drivers.*

Note
This book uses the abbreviated term *protocol* to refer to the implementation of those rules in software drivers, instead of always using the complete term *protocol implementation.* ◆

There are many ways to characterize networks. One characteristic of a network is whether it is connection-oriented or connectionless. (A protocol can also be considered connectionless or connection-oriented.) A connection-oriented network is one in which

two nodes on the network, such as computers, that want to communicate must go through a connection-establishment process, which is called a *handshake.* This involves the exchange of predetermined signals between the nodes in which each end identifies itself to the other. Once a connection is established, the communicating applications or processes on the nodes at either end can send and receive streams of data.

A ***connectionless network*** is one in which two nodes that want to communicate do so by going directly into a data-transfer state without first setting up a connection. A connectionless network is also called a ***datagram*** or ***packet-oriented network*** because data is sent as discrete packets; a ***packet*** is a small unit of data that is sent across a network. This means that each packet must carry the full addressing information required to deliver the data from its source node to its destination node. A packet includes a ***header*** portion that holds the addressing information along with some other information, such as a checksum value that can be used to verify the integrity of the data delivered, and a data portion that holds the message text. The terms *packet* and *datagram* are synonymous.

A connection-oriented network is analogous to a telephone system. The party who initiates the call knows whether or not the connection is made because someone at the other end of the line either answers or not. A connectionless network is analogous to electronic mail. A person sends a mail message expecting it will be delivered to its destination. Although the mail usually arrives safely, the sender doesn't know this unless the recipient initiates a response affirming it.

There are trade-offs between the two types of networks: a connection-oriented network provides more function, but at a cost. A connectionless network is less costly in terms of overhead, but it offers limited support.

A connection-oriented network ensures ***reliable delivery of data,*** which includes error checking and recovery from error or packet loss. Connection-oriented networks provide support for sessions. In AppleTalk networking, a ***session*** is a logical (as opposed to physical) connection between two entities on an internet. The two communicating parties can send streams of data across a session, rather than being limited to sending the data as individual packets. When data is sent as a stream, the networking system provides flow control to manage the data that makes up the stream. A session must be set up at the beginning and broken down at the end. All of these services entail overhead.

There is no connection setup or breakdown required for a connectionless network, and no session is established. A connectionless network offers best-effort delivery only. ***Best-effort delivery*** means that the network attempts to deliver any packets that meet certain requirements, such as containing a valid destination address, but the network does not inform the sender when it is unable to deliver the packet, nor does it attempt to recover from error conditions and packet loss. A connectionless network involves less overhead because it does not provide network-wide acknowledgments, flow control, or error recovery.

The terms *connectionless* and *connection-oriented* can also be applied to individual protocols that make up the networking software, as well as to the entire network system itself. AppleTalk includes protocols that provide connection-oriented services, although, as a whole, AppleTalk is considered a connectionless network because data is delivered

across an AppleTalk network or internet as discrete packets. One of the AppleTalk protocols, the Datagram Delivery Protocol (DDP), implements packet delivery. However, the AppleTalk Data Stream Protocol (ADSP) and the AppleTalk Transaction Protocol (ATP) provide connection-oriented services, such as session establishment and reliable delivery of data. The AppleTalk protocols that provide connection-oriented services are built on top of the datagram services that DDP provides.

In developing AppleTalk applications, you must decide whether to use a connection-oriented or connectionless AppleTalk protocol. How to choose a protocol to use is described in "Deciding Which AppleTalk Protocol to Use" beginning on page 1-22.

The connection-oriented AppleTalk protocols support the following two kinds of sessions:

- ***symmetrical.*** This session is also referred to as a *peer-to-peer session.* It is one in which both ends have equal control over the communication. Both ends can send and receive data at the same time and initiate or terminate the session. This type of session offers more capability and is more commonly used than an asymmetrical session.
- ***asymmetrical.*** In this type of session, only one end of the connection can control the communication. One end of the connection makes a request to which the other end can only respond. This type of session is best suited to a transaction in which a small amount of data is transferred from one side to the other.

When both ends can send and receive data, the process is called a ***full-duplex dialog.*** When both sides must alternate between sending and receiving data, the process is called a ***half-duplex dialog.***

Addressing and Data Delivery on AppleTalk Networks

This section discusses some of the aspects of AppleTalk networking that are part of its addressing and data-delivery scheme. Many components contribute to the addressing information that is used to identify the location of an application or a process on an AppleTalk internet. This section defines these names and numbers, and Table 1-1 highlights them.

Table 1-1 AppleTalk addressing numbers and names

Addressing information	Description
Network number	A unique 16-bit number that identifies the network to which a node is connected. A single AppleTalk network can be either extended or nonextended. An extended network is defined by a range of network numbers.
Node ID	A unique 8-bit number that identifies a node on an AppleTalk network.
Socket number	A unique 8-bit number that identifies a socket. A maximum of 254 different socket numbers can be assigned in a node.
Zone name	A name assigned to an arbitrary subset of nodes within an AppleTalk internet.

A single AppleTalk network can be interconnected with other AppleTalk networks through ***routers*** to create a large, dispersed AppleTalk internet. A router in an internet can select the most efficient path to the data's intended destination, while allowing connected networks to remain fully independent and to retain separate addresses.

Each network is assigned a ***network number*** so that packets destined for a particular network on an AppleTalk internet can be routed to the correct network. A router consults the packet's destination network number and forwards the packet throughout the internet from one router to another until the packet arrives at its destination network. AppleTalk supports a number of types of networks including LocalTalk, TokenTalk, EtherTalk, and FDDITalk networks.

AppleTalk assigns a ***node ID*** to a node when it connects to the network. Every node on an AppleTalk network is identified by its unique 8-bit node ID. (Extended networks include the 16-bit network number.) Once a packet arrives at its destination network, the packet is delivered to its destination node within that network, based on the node ID.

More than one application or process that uses AppleTalk may be running on a single node at the same time. Because of this, AppleTalk must have a way to determine for which application or process a packet that is delivered to the node is intended. AppleTalk uses sockets to satisfy this requirement. A ***socket*** is a piece of software that serves as an addressable entity on a node. Each process or application that runs on an AppleTalk network "plugs into" a socket that is identified by a unique number. Applications or processes exchange data with each other across an internet through sockets. Because each application or process has its own socket address, a node can have two or more concurrent open connections, for example, one to a file server and one to a printer.

The ***socket number*** identifies the process to which the Datagram Delivery Protocol (DDP) is to deliver a packet. The combination of the socket number, the node ID, and the network number creates the ***internet socket address*** of an application or process. An internet socket address provides a unique identifier for any socket in the AppleTalk internet. When an application or process is associated with a socket, it is referred to as a ***socket client.***

An application or process becomes accessible from any point in the AppleTalk internet through its association with an internet socket address and a special name that is associated with the internet socket address through the AppleTalk Name-Binding Protocol (NBP). An NBP name contains three parts: object, type, and zone. The zone field of the name is the zone in which the node resides.

A ***zone*** is a logical grouping of nodes in an AppleTalk internet. The use of zones allows a network administrator to set up departmental or other logical groupings of nodes on an internet. A single extended network can contain nodes belonging to any number of zones; an individual node on an extended network can belong to only one zone. Each zone is identified by a zone name.

An ***AppleTalk internet*** always consists of more than one AppleTalk network. It can be made up of a mix of LocalTalk networks, TokenTalk networks, EtherTalk networks, and FDDITalk networks. It can also consist of more than one network of a single type, such as

several LocalTalk networks. A single AppleTalk network can be either a nonextended network or an extended network. An AppleTalk internet can include both nonextended and extended networks.

Note
The term *internet* is used throughout this book to refer to an AppleTalk internet exclusively. It is not within the scope of this book to discuss other types of internets, such as Arpanet. ◆

An AppleTalk ***nonextended network*** is one in which

- the network has one network number assigned to it
- the network supports only one zone
- all nodes on the network share the same network number and zone name
- each node on the network has a unique node ID

LocalTalk is an example of a nonextended network. Each node on a nonextended network, such as LocalTalk, has a unique 8-bit node ID. Because there are 256 combinations of 8 bits, and two combinations are not available (ID 255 is reserved for broadcast messages and the ID 0 is not allowed), a nonextended network supports up to only 254 active nodes at a time.

An AppleTalk ***extended network*** is one in which

- the network has a range of network numbers assigned to it
- the network supports multiple zones
- each node on the network has a unique node ID (Nodes can also have different network numbers that fall within the network number range and different zone names.)

A network number range defines the extended network. An extended network uses what is referred to as ***extended addressing:*** in principle, a range of network numbers allows each extended network to have over 16 million (2^{24}) nodes. In any specific implementation, the hardware or software might limit the network to fewer nodes.

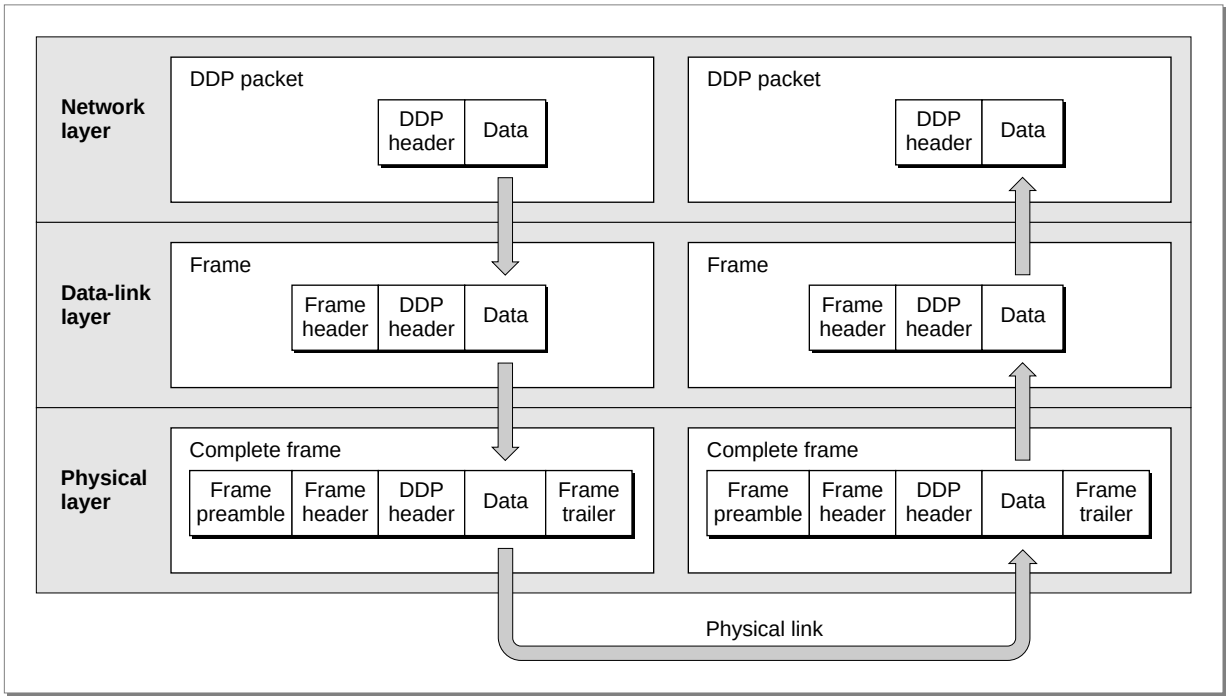
You can think of an extended network as a number of nonextended networks forming a single network, each providing up to 254 possible node IDs.

Whether the network is extended or nonextended, data is always delivered in DDP packets that include the DDP header that contains addressing information followed by the data itself. As the DDP packet is passed down the protocol stack to the layer below, the packet is extended to include additional information.

At the data-link layer, additional addressing information is prepended to the DDP header, and the packet is now called a *frame.* At the physical layer, a frame preamble is prepended to the frame header and a frame trailer is appended to the end of the data portion of the DDP packet. (You don't need to be concerned with the frame preamble and frame trailer; they are mentioned here and shown in Figure 1-1 for completeness.) The frame is then transmitted across the network or internet to its destination node.

At the destination node, the frame is received, and as it is passed up through the protocol stack the additional information that was added to the DDP packet at each layer on the sending node is used and removed at the corresponding layer on the destination node. The frame preamble and frame trailer are removed at the physical layer. The frame header is removed at the data-link layer. You can think of the data that your application sends as being enclosed successively at each of these layers in envelopes that contain addressing information necessary to deliver the data; at the corresponding layer on the destination node, the envelope is removed. Figure 1-1 illustrates this concept.

Figure 1-1 Data delivery on AppleTalk networks



AppleTalk Connectivity

A fundamental part of a network system is its ***connectivity*** infrastructure, which includes the communication hardware and the protocols for controlling the hardware. The communication hardware can consist of various media including wire cabling, fiber optics cabling, and a network interface controller (NIC), if one is used. This hardware and software constitute the data transmission medium, which is called a data link. A data link provides nodes with access to the network.

Nodes on a network share and compete for access to the link. The link-access protocol implemented in the software controls the access of a node to the network hardware and makes it possible for many nodes to share the same communications hardware. It also handles the delivery of packets from one node to another over a network. When a packet

is delivered to the link-access protocol for transmission across the network, additional addressing and control information is added to the packet, and the packet is called a ***frame.***

AppleTalk connectivity is designed to be ***link independent,*** which means that it allows for the use of various types of data links accessed through the various link-access protocols, which it supports. AppleTalk provides the following data-link support:

- The LocalTalk Link-Access Protocol (LLAP) supports a LocalTalk link.
- The EtherTalk Link-Access Protocol (ELAP) supports an Ethernet link.
- The TokenTalk Link-Access Protocol (TLAP) supports a token ring link.
- The Fiber Distributed Data Interface Link-Access Protocol (FLAP) supports a Fiber Distributed Data Interface link.

These protocols provide interfaces between the Datagram Delivery Protocol (DDP) and the types of data-link hardware that AppleTalk can use. A user can choose to connect to any of the data links that the node is set up to support.

AppleTalk includes a component called the ***Link-Access Protocol (LAP) Manager,*** which insulates the higher-level AppleTalk protocols from having to identify and connect to the link that the user has chosen; the LAP Manager connects to the selected link for them.

AppleTalk Phase 2

The current version of AppleTalk, which was introduced in 1989, is AppleTalk Phase 2. Based on the original version of AppleTalk, it was designed to enhance performance over large networks through the following improvements:

- The routing protocols that specify how messages are passed between networks were enhanced to promote improved network traffic and better router selection.
- Extended addressing, which allows a range of network numbers to be assigned to a single network, was implemented for networks other than LocalTalk.
- Support of multiple zones for extended networks was added. An extended network can have an associated list of zone names. A single extended network can be associated with more than one zone name, or a single zone name can be associated with more than one extended network. Two nodes on the same extended network can belong to different zones.

Note

The Phase 2 versions of the AppleTalk drivers are included as part of system software version 7.0 and later. They can be installed on any Macintosh computer other than the Macintosh 128K, Macintosh 512K, Macintosh 512K enhanced, and Macintosh XL computers. If you want to provide AppleTalk Phase 2 drivers with your product, you must obtain a license from Apple Software Licensing. ◆

Historical note

AppleTalk Phase 1, the original AppleTalk protocol architecture, was designed to support small local workgroups. AppleTalk Phase 1 supported the LocalTalk Link-Access Protocol (LLAP), which was originally called the *AppleTalk Link-Access Protocol (ALAP)*. With the addition of the EtherTalk Link-Access Protocol (ELAP) and other link-access protocols, ALAP was renamed to indicate the specific data link that it supports. ◆

The AppleTalk Protocol Stack

This section explains what an AppleTalk protocol is, then it provides a brief discussion of each component of the AppleTalk protocol stack, followed by a discussion of how the AppleTalk protocols are implemented in software drivers.

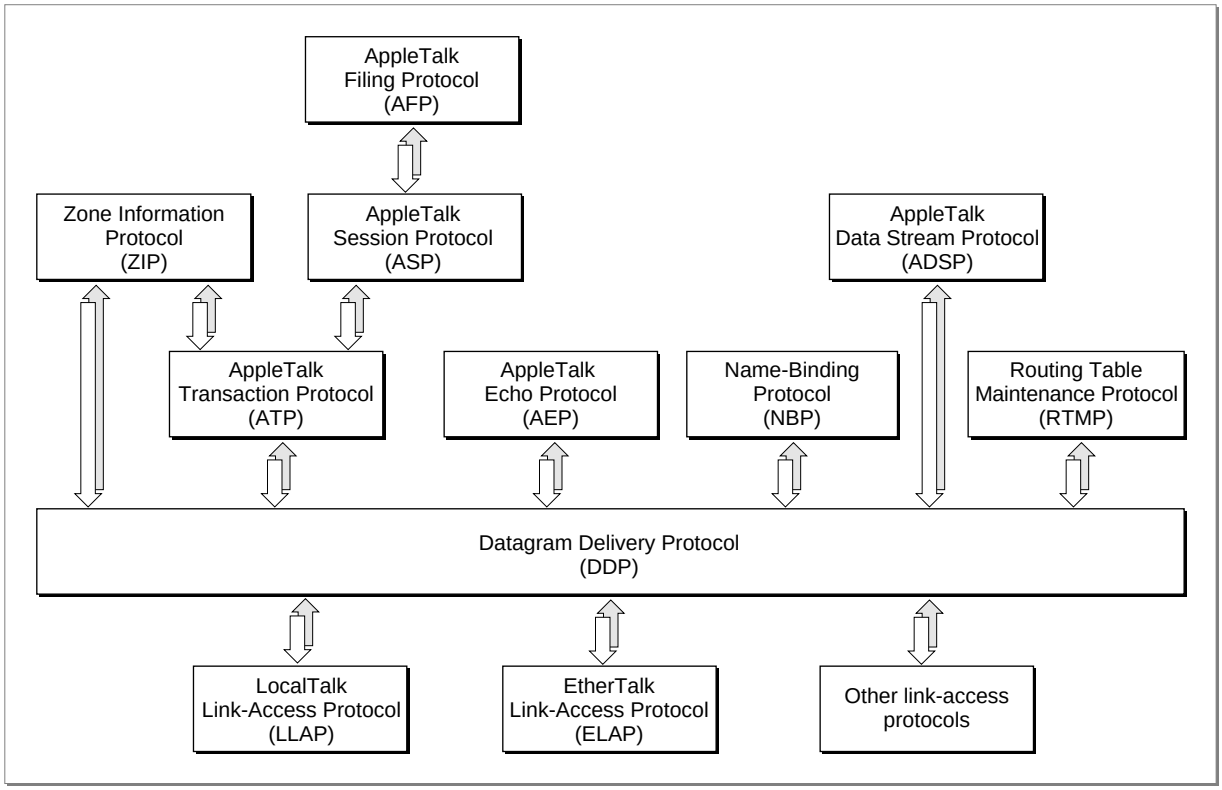
This section also introduces the LAP Manager, multivendor support, and multinode architecture, which are components of AppleTalk, although strictly considered, they are not protocols.

To develop applications that use AppleTalk networking services, you don't need to understand how AppleTalk implements the protocols it supports. However, understanding the functions that each protocol provides will help you determine which application programming interfaces to use for your application.

The AppleTalk system architecture consists of a number of protocols arranged in layers. The various AppleTalk protocols are sets of rules, not computer programs, and so can be implemented in many different ways on many different systems. All of the AppleTalk protocol functions that you can address or control from a Macintosh application are implemented as Macintosh device drivers or managers. Many other features of these protocols are implemented in software located only on internet routers that are not used to run general applications. Some parts of protocols are implemented by server software such as file servers or print servers.

When this book refers to a protocol as *doing* or *controlling* something, you should understand the statement to mean that some program that implements the protocol actually carries out the operation. Each protocol in a specific layer provides services to one or more protocols in a higher-level layer, which is then the client of the lower-level protocol. The higher-level protocol builds on the services provided by the lower-level one. Figure 1-2 on page 1-12 shows the AppleTalk protocols and how they relate to one another in layers. The following sections describe each protocol in turn, beginning with AFP, and progressing through the protocols as they appear in the figure.

Figure 1-2 AppleTalk protocol stack



AppleTalk Filing Protocol (AFP)

The ***AppleTalk Filing Protocol (AFP)*** allows a workstation on an AppleTalk network to access files on AppleShare file servers. When the user opens a session with an AppleShare file server over an internet, it appears to any application running on the workstation that uses File Manager routines as if the files on the file server were located on a disk drive connected to the workstation. The AFP protocol is not commonly used because the native file system commands allow users to access an AFP server, such as AppleShare, from a workstation. There is no server-based interface.

The chapter "AppleTalk Filing Protocol (AFP)" in this book describes the application programming interface to the workstation implementation of AFP. For additional information about AFP, see "Accessing AppleShare and Other File Servers" on page 1-27.

Zone Information Protocol (ZIP)

The ***Zone Information Protocol (ZIP)*** provides applications and processes with access to zone names. Each node on a network belongs to a zone. Zone names are typically used

to identify groups of nodes belonging to a particular department or area. ZIP allows applications and processes to gain access to

- their own node's zone name
- the names of all the zones on their local network
- the names of all the zones throughout the internet

The chapter "Zone Information Protocol (ZIP)" in this book describes the ZIP application programming interface. For additional information about ZIP, see "Identifying Zones" on page 1-23.

AppleTalk Session Protocol (ASP)

The ***AppleTalk Session Protocol (ASP)*** sets up and maintains sessions between a workstation and a server. ASP is an asymmetrical protocol in which one side of the dialog, the workstation client of ASP, initiates the session and sends commands to the other side of the dialog. A higher-level protocol that is built on top of the ASP server interprets and executes the command, and the ASP server returns a reply. ASP also provides a means by which the server can send a message to the workstation; for example, a file server can use this messaging system to notify all of the workstations that are using the file server that it is shutting down. ASP is used by the AppleTalk Filing Protocol to allow a user to manipulate files on a file server. Because ADSP provides socket clients at both ends of the connection with equal control, ADSP is more commonly used than ASP when a session protocol is required.

The chapter "AppleTalk Session Protocol (ASP)" in this book describes the ASP application programming interface. For additional information about ASP, see "AppleTalk Session Protocol" on page 1-25.

AppleTalk Data Stream Protocol (ADSP)

The ***AppleTalk Data Stream Protocol (ADSP)*** is a connection-oriented protocol that supports sessions over which applications and processes that are socket clients can exchange full-duplex streams of data across an AppleTalk internet. ADSP is a symmetrical protocol; the socket clients at either end of the connection have equal control over the ADSP session and the data exchange. Through attention messages, ADSP also provides for out-of-band signaling, a process of sending data outside the normal session dialog so as not to interrupt the data flow.

The chapter "AppleTalk Data Stream Protocol (ADSP)" in this book describes the ADSP application programming interface. For additional information about ADSP, see "AppleTalk Data Stream Protocol" on page 1-24.

AppleTalk Transaction Protocol (ATP)

The ***AppleTalk Transaction Protocol (ATP)*** is a transaction protocol that allows one socket client to transmit a request that some action be performed to another socket client that carries out the action and transmits a response reporting the outcome. ATP provides reliable delivery of data by retransmitting any data packets that are lost and ensuring that the data packets are delivered in the correct sequence.

The chapter "AppleTalk Transaction Protocol (ATP)" in this book describes the ATP application programming interface. For additional information about ATP, see "Performing a Transaction" on page 1-25.

AppleTalk Echo Protocol (AEP)

The ***AppleTalk Echo Protocol (AEP)*** exists on every node as a DDP client process called the ***AEP Echoer.*** The AEP Echoer uses a special socket to *listen* for packets sent to it from socket clients on other nodes. When it receives such a packet, the AEP Echoer returns it directly to the sender. A socket client can send a packet to the AEP Echoer on another node to determine if that node can be accessed over the internet and to determine how long it takes a packet to reach that node. There is no application programming interface to AEP. A socket client can send packets to an AEP Echoer socket on another node from a DDP socket, but it cannot access the AEP implementation directly.

The chapter "Datagram Delivery Protocol (DDP)" in this book describes how to send packets to the AEP socket. For additional information about AEP, see "Measuring Packet-Delivery Performance" on page 1-26.

Name-Binding Protocol (NBP)

The ***Name-Binding Protocol (NBP)*** provides your application or process with a way to map names that are useful to people using your program to numbers or addresses that are useful to computers. NBP associates a user-friendly three-part name that can be displayed to end users with the internet socket address of the application or process. When a user launches it, your application can register itself with NBP. When a user quits the application or when you no longer wish to advertise your application, your application can delete its entry from the NBP names table. Once your application registers itself with NBP, other applications can locate it.

All applications and processes that use AppleTalk use NBP to make their services known and available throughout an AppleTalk internet and to locate other applications and processes in the internet. An application or process can use NBP to

- register itself with NBP. Registering an application or process with NBP makes that process a network-visible entity. (NBP lets your application or process bind a three-part name to its internet socket address.)
- look up or confirm the address of another application or process that is registered with NBP.
- remove its entry from the NBP names table when it no longer wants to advertise its services.

The chapter "Name-Binding Protocol (NBP)" in this book describes the NBP application programming interface. For additional information about NBP, see "Making Your Application Available Throughout the Internet" on page 1-22.

Routing Table Maintenance Protocol (RTMP)

The ***Routing Table Maintenance Protocol (RTMP)*** provides AppleTalk internet routers with a means of managing routing tables used to determine how to forward a datagram from one socket to another across an internet based on the datagram's destination network number. The RTMP implementation on a router maintains a table called a routing table that specifies the shortest path to each possible destination network number. The AppleTalk protocol software in a workstation (that is, a node other than a router) contains only a small part of RTMP, called the RTMP stub, that DDP uses to determine the network number (or range of network numbers) of the network cable to which the node is connected and to determine the network number and node ID of one router on that network cable. There is no application programming interface to the RTMP stub; therefore, RTMP is not discussed in this book.

Datagram Delivery Protocol (DDP)

The ***Datagram Delivery Protocol (DDP)*** is a connectionless protocol that transfers data between sockets as discrete packets, or datagrams, with each packet carrying its destination internet socket address. DDP provides best-effort delivery. It does not include support to ensure that all packets sent are received at the destination or that those packets that are received are in the correct order. Higher-level protocols that use the services of DDP provide for reliable delivery of data. DDP uses whichever link-access protocol the user selects; that is, DDP can send its datagrams through any type of data link and transport media.

The chapter "Datagram Delivery Protocol (DDP)" in this book describes the DDP application programming interface. For additional information about DDP, see "Sending and Receiving Data as Discrete Packets" on page 1-26.

Link-Access Protocols

AppleTalk supports various network (or link) types and allows the user to select and switch among the types of networks to be used based on how the user's machine is configured; that is, if the machine has the proper hardware and software installed for a link type, the user can select that link. AppleTalk includes the link-access protocols for LocalTalk, EtherTalk, TokenTalk, and FDDITalk (Fiber Distributed Data Interface). AppleTalk uses connection files of type `'adev'` that contain software that supports a particular type of data link.

To achieve link independence, AppleTalk relies on the ***Link-Access Protocol (LAP) Manager,*** which is a set of operating-system utilities, not an AppleTalk protocol. The main function of the LAP Manager is to act as a switching mechanism that connects the AppleTalk link-access protocol for the link type that the user selects to both the higher-level AppleTalk protocols and the lower-level hardware device driver for that data link. From the Network control panel, a user can select which network is to be used for the node's AppleTalk connection.

The AppleTalk connection files of type `'adev'` and the LAP Manager work together with the Network control panel file of type `'cdev'`. When the user selects a network type from the Network control panel, the LAP Manager routes AppleTalk communications through the link-access protocol for the selected network.

The LAP Manager also provides an application with access to the AppleTalk Transition Queue. You can place an entry for your application in the AppleTalk Transition Queue so that the LAP Manager will notify you when an AppleTalk transition occurs or is about to occur. An AppleTalk transition is an event, such as an AppleTalk driver being opened or closed, that can affect your AppleTalk application.

The chapter "Link-Access Protocol (LAP) Manager" in this book describes the LAP Manager and the AppleTalk Transition Queue. For additional information about the LAP Manager, see the *Macintosh AppleTalk Connections Programmer's Guide*.

Multivendor Architecture

In addition to supporting various types of networks, Apple also provides what is known as *multivendor support*. The ***multivendor architecture*** allows for multiple brands of Ethernet, token ring, and FDDI NuBus™ network interface controllers (NICs) to be installed on a single node at the same time. In addition to selecting the type of network connection, the user can now select a particular device to be used for the network connection. The chapter "Ethernet, Token Ring, and Fiber Distributed Data Interface" in this book describes multivendor architecture.

Multinode Architecture

Multinode architecture is an AppleTalk feature that allows an application to acquire node IDs in addition to the standard node ID that is assigned to the system when the node joins an AppleTalk network. Multinode architecture is provided to meet the needs of special-purpose applications that receive and process AppleTalk packets in a custom manner instead of passing them directly on to a higher-level AppleTalk protocol for processing. A multinode ID allows the system that is running your application to appear as multiple nodes on the network. The prime example of a multinode application is Apple Remote Access (ARA). The chapter "Multinode Architecture" in this book describes this feature.

How the AppleTalk Protocols Are Implemented

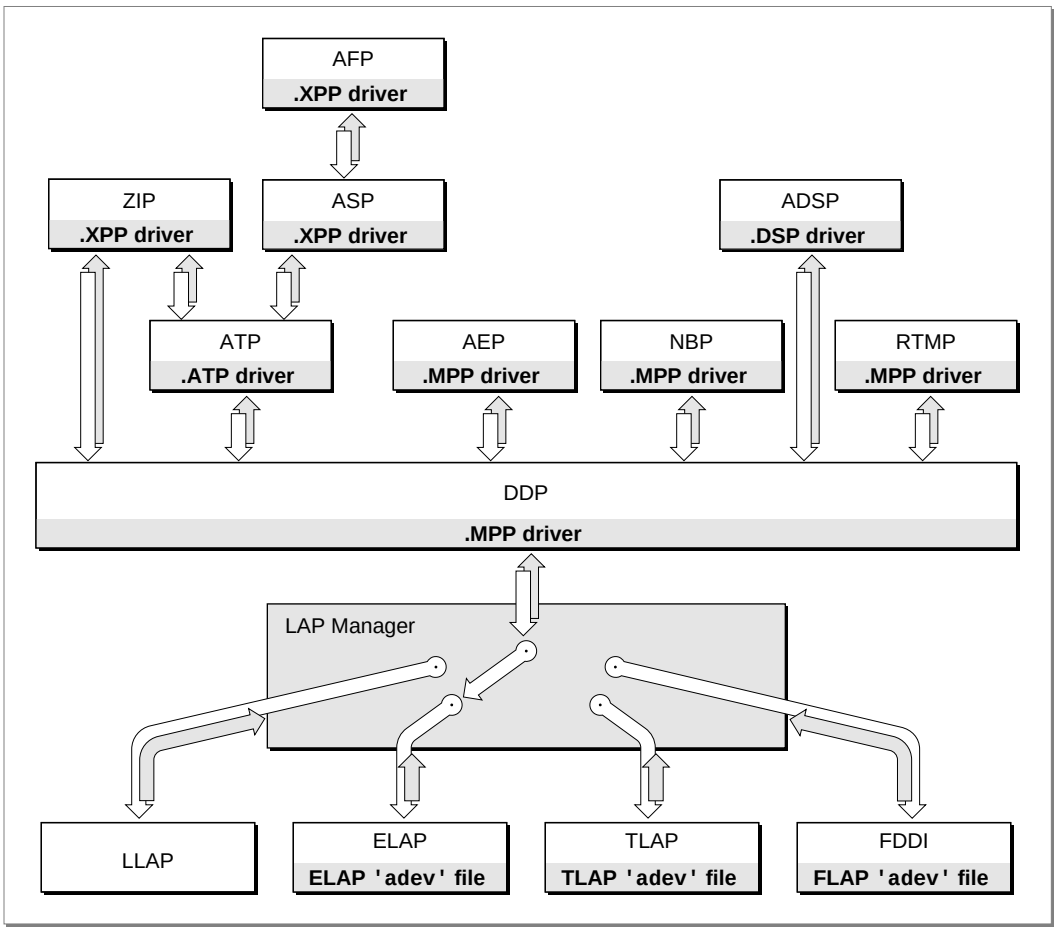
Above the data-link level, all of the AppleTalk protocols that you can address or control from a Macintosh application through a programming interface as well as multinode architecture are implemented as Macintosh device drivers. Table 1-2 identifies the AppleTalk drivers and the protocols they implement.

Table 1-2 AppleTalk drivers and the protocols they implement

AppleTalk driver	Protocols it implements
.MPP	DDP, NBP, AEP, RTMP stub, multinode
.ATP	ATP
.XPP	ASP, workstation portions of ZIP and AFP
.DSP	ADSP

Figure 1-3 shows the AppleTalk protocols with the name of the driver that implements the protocol and the connection files of type `'adev'` that AppleTalk provides for various types of links. Notice how the LAP Manager acts as a switching mechanism between the higher-level protocols and the link-access protocols. Many other features of these protocols are implemented in software located only on internet routers that are not used to run general applications. Some parts of protocols are implemented by server software such as file servers and print servers.

Figure 1-3 Device drivers and connections files that implement AppleTalk protocols

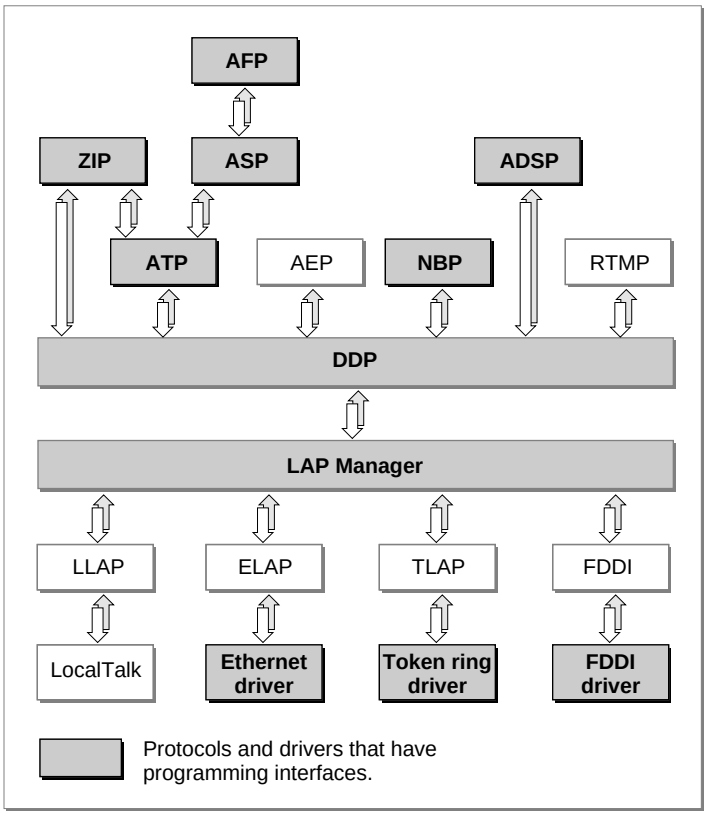


The AppleTalk Manager

Your application accesses the services of the AppleTalk protocols through the AppleTalk Manager, which is a collection of the application programming interfaces to the AppleTalk protocols. The AppleTalk Manager includes the LAP Manager, which collects together the interfaces to the supported AppleTalk data links. Note that not all AppleTalk protocols have programming interfaces.

Figure 1-4 shows the AppleTalk protocols; those protocols that have programming interfaces are shaded.

Figure 1-4 AppleTalk protocols with programming interfaces



Typically, an application uses the services of more than one protocol. For example, you might choose to use ADSP to set up a symmetrical session over which the users of your application can transfer data, but you would also use NBP to register your application to make it available to users and other applications throughout the internet. For information on how to select which protocols to use, see "Deciding Which AppleTalk Protocol to Use" on page 1-22.

AppleTalk and the OSI Model

This section provides general information about the relationship between AppleTalk and an industry-standard networking model. You do not need to read this section to understand the AppleTalk protocols or to use the AppleTalk Manager.

Most networking systems are designed as layered architectures that relate to what are called *reference models*. These matrices offer a structure that network designers can refer to in developing a network architecture; they are guidelines and not rules. Each layer of a model collects together those functions that are similar or highly interrelated and provides services to the layer above it. Network designers develop protocols that encompass the functions of each layer. Often more than one protocol is defined and implemented to handle the requirements of a layer in different ways. Some protocols include functions that span more than one layer specified by a model. For example, in favor of efficiency, a network protocol developer may elect to define a single protocol that spans two or more layers of a reference model.

Various layered models have been developed that provide standards for the design and development of networking software. One of these models is the Open Systems Interconnection (OSI) model, which is a seven-layered standard that was published by the International Standards Organization (ISO) in the 1970s. This is the model with which the AppleTalk network system architecture is most closely aligned.

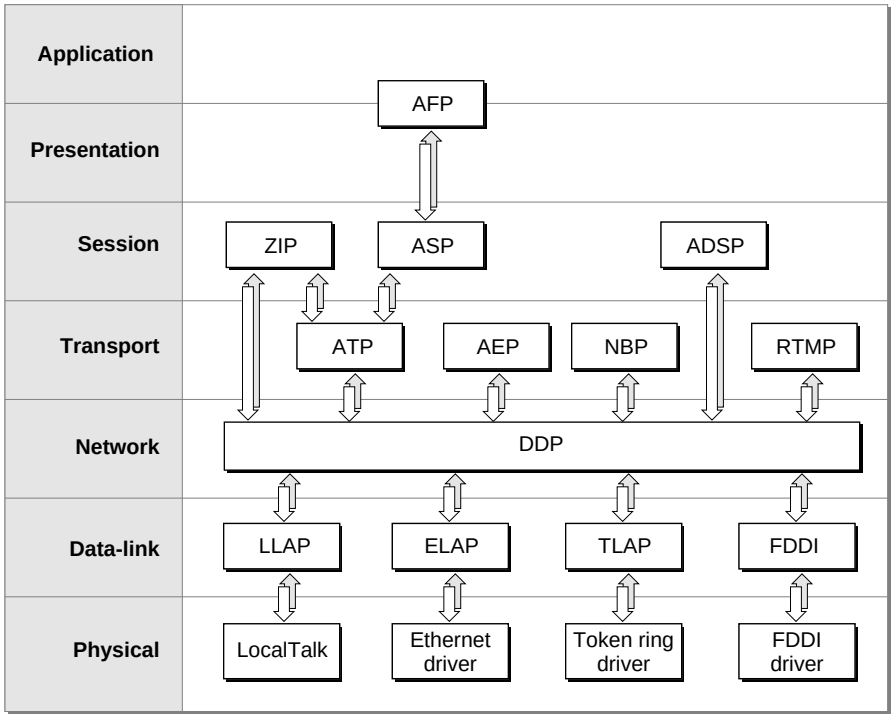
Note
Although this section discusses AppleTalk in relation to the OSI model, it does not claim a protocol compatibility of AppleTalk with the OSI protocols currently in various stages of definition, approval, and deployment. ◆

Figure 1-5 on page 1-20 shows the relationships among the AppleTalk protocols and how they map to the OSI model. The shaded area of the graphic shows the name of the OSI layer. A connection between one protocol and another above or below it in the figure indicates that the upper protocol is a client of the lower protocol, that is, the upper protocol uses services provided by the lower protocol in order to carry out some functions.

Application Layer

The highest layer of the OSI model is the application layer. This layer allows for the development of application software. Software written at this layer benefits from the services of all the underlying layers. There is no AppleTalk protocol that maps directly to this layer, although some of the functions of the AppleTalk Filing Protocol (AFP) fulfill this layer.

Figure 1-5 AppleTalk protocol stack and the OSI model



Presentation Layer

The presentation layer assumes that an end-to-end path or connection already exists across the network between the two communicating parties, and it is concerned with the representation of data values for transfer, or the *transfer syntax.* In the OSI model, the AppleTalk Filing Protocol (AFP) spans the presentation and application layers. AFP provides an interface between an application and a file server. It uses the services of ASP, which, in turn, is a client of ATP.

AFP allows a workstation on an AppleTalk network to access files on an AFP file server, such as an AppleShare file server. When the user opens a session with an AppleShare file server over an internet, it appears to any application running on the workstation that uses File Manager routines as if the files on the file server were located on a disk drive connected to the workstation.

Session Layer

The session layer serves as an interface into the transport layer, which is below it. The session layer allows for ***session establishment,*** which is the process of setting up a connection over which a dialog between two applications or processes can occur. Some of the functions that the session layer provides for are flow control, establishment of synchronization points for checks and recovery for file transfer, full-duplex and half-duplex dialogs between processes, and aborts and restarts.

The AppleTalk protocols implemented at the session layer are

- the AppleTalk Data Stream Protocol (ADSP), which provides its own stream-based transport layer services that allow for full-duplex dialogs
- the AppleTalk Session Protocol (ASP), which uses the transaction-based services of ATP to transport workstation commands to servers
- the Zone Information Protocol (ZIP), which provides applications and processes with access to zone names. Each node on a network belongs to a zone.

Transport Layer

The transport layer isolates some of the physical and functional aspects of a packet network from the upper three layers. It provides for end-to-end accountability, ensuring that all packets of data sent across the network are received and in the correct order. This is the process that is referred to as *reliable delivery of data*, and it involves providing a means of identifying packet loss and supplying a retransmission mechanism. The transport layer also provides connection and session management services.

The following AppleTalk protocols are implemented at the transport layer:

- Name-Binding Protocol (NBP)
- AppleTalk Transaction Protocol (ATP)
- AppleTalk Echo Protocol (AEP)
- Routing Table Maintenance Protocol (RTMP)

In addition to these transport layer protocols, the AppleTalk Data Stream Protocol (ADSP) includes functions that span both the transport and the session layers. ADSP provides for reliable delivery of data, and in that capacity it covers the transport layer requirements.

Network Layer

The network layer specifies the network routing of data packets between nodes and the communications between networks, which is referred to as *internetworking*. The Datagram Delivery Protocol (DDP) is the AppleTalk protocol implemented at the network layer. DDP is a connectionless datagram protocol providing best-effort delivery. This means that DDP transfers data as discrete packets and that DDP does not include support to ensure that all packets sent are received at the destination or that those packets that are received are in the correct order. Higher-level protocols that use the services of DDP provide for this kind of reliability.

Data-Link and Physical Layers

The data-link layer and the physical layer provide for connectivity. The communication between networked systems can be via a physical cable made of wire or fiber optic, or it can be via infrared or microwave transmission. In addition to these, the hardware can include a network interface controller (NIC), if one is used. The hardware or transport media and the device drivers for the hardware comprise the physical layer. LocalTalk,

token ring, Ethernet, and Fiber Distributed Data Interface (FDDI) are examples of types of networking hardware that AppleTalk supports.

The physical hardware provides nodes on a network with a shared data transmission medium called a *link*. The data-link layer includes a protocol that specifies the physical aspects of the data link and the link-access protocol, which handles the logistics of sending the data packet over the transport medium. AppleTalk is designed to be data-link independent, allowing for the use of various types of hardware and their link-access protocols.

Deciding Which AppleTalk Protocol to Use

The AppleTalk Manager consists of a collection of application programming interfaces to the AppleTalk protocols and the LAP Manager. Each of the AppleTalk protocols implements a different set of functions and services, and the programming interface for a specific protocol includes a set of routines that give your application access to the protocol's functions and services.

AppleTalk offers programming interfaces to a variety of communications protocols at different levels. Your choice of protocol or protocols to use depends primarily on your application's needs.

This section provides a brief discussion of how your application can use each protocol. The AppleTalk protocols are layered in a stack with each protocol benefiting from the services of the protocols in layers below it. Looked at from a top-down approach, the high-level protocols provide an accretion of all the services of the underlying protocols.

A developer who uses the higher-level protocols that provide for reliable delivery of data and error recovery does not have to implement these services as part of an application. An application developer who wants to write a program for end users that runs on an AppleTalk network would typically use the interfaces to one or more higher-level protocols. For example, you might use NBP to register the program with the network so that it is visible to users and other applications, and, perhaps, ADSP to transfer data.

A network software developer who wants to implement a custom session-oriented protocol, instead of using ADSP or ASP, would typically use the interface to a protocol such as DDP or any of the protocols below it. A network software developer who wants to implement a custom protocol stack instead of using AppleTalk can use a low-level protocol interface to attach a protocol handler that receives data from the network.

Making Your Application Available Throughout the Internet

This section discusses the Name-Binding Protocol (NBP) that you can use to make your application or process visible to users and other applications and processes throughout an AppleTalk internet.

NBP binds the internet socket address assigned to a process or application to a special human-readable name that contains three parts: the object, type, and zone fields. The NBP name is different from the name of the application. The object and type are assigned by the process itself and can be anything the user or application developer selects; the zone is the one in which the node resides.

NBP maintains a table on each node that contains the name-and-address pair for each application or process on that node that is registered with NBP. Once an application or process is registered with NBP, it becomes visible to users and other applications and processes throughout the internet. When a process or application is registered with NBP, it is referred to as a ***network-visible entity.***

Users can select an application by its NBP name. Based on the name or a part of the name, applications and processes can request NBP to look up the internet socket address for the entity.

When you use other AppleTalk protocols that send and receive data, your application or process becomes associated with an internet socket address. Although applications and processes need the internet socket addresses of other applications and processes that they want to connect with, a name identifying the type of application and its location is more meaningful to an end user. Your application or process can use NBP to find all other applications or processes of the same type and get their internet socket addresses. Your application could then display the NBP names of other applications to an end user so that the user can select an application to connect to. Your application could then use another AppleTalk protocol, such as ADSP, to connect to the partner application.

An application, such as a network management tool, could use NBP to collect information so that it can provide an inventory of all nodes belonging to a zone and list the applications running on each of those nodes. It could sort the applications by type. For example, it could provide a list of all file servers on an AppleTalk internet.

Identifying Zones

The Zone Information Protocol (ZIP) maintains a zone information table in each internet router that lists the relationships between zone names and network numbers. You can use the part of ZIP that is implemented on a nonrouter node to get the name of the zone to which the node that is running the application belongs. Your application can also call ZIP to get a list of all the zones in the internet.

An application running on a node that belongs to an extended network can call ZIP to get a list of all the zone names associated with that network. For example, an application that supports network administration might use these service to provide a network administrator with a list of the zones for a particular network so that the administrator can select the correct zone for a node when adding nodes to a network.

An application could collect other kinds of information, such as what services are running on nodes, and then sort the information by zone.

Using a Session Protocol to Send and Receive Data

AppleTalk includes two session protocols that you can use to send and receive data:

- ADSP provides a symmetrical session.
- ASP provides an asymmetrical session.

Most applications use ADSP, which was made available after ASP.

AppleTalk Data Stream Protocol

Your application can use ADSP to set up and maintain a connection with another application over an internet. Through this connection, both applications can send and receive streams of data at any time. Because ADSP allows for the continuous exchange of data, any application that needs to support the exchange of more than a small amount of data should use ADSP. In addition to providing for a duplex data stream, ADSP also provides an application with a means of sending attention messages to pass control information between the two communicating applications without disrupting the main flow of data.

In most cases, ADSP is the protocol that Apple recommends applications use for sending and receiving data. In addition to ensuring reliable delivery of data, ADSP provides a peer-to-peer connection, that is, both ends of the connection can exert equal control over the exchange of data.

Note
Because ADSP is connection-oriented, it entails additional processing and memory usage in setting up and maintaining the connection between the two applications. Therefore, if your application needs to send a small amount of data, such as a request that the other end perform a task and report the result in response, and you don't want to incur the overhead involved in establishing, maintaining, and breaking a connection, you should consider using ATP rather than ADSP for data transfer. ◆

ADSP appears to its clients to maintain an open pipeline between the two entities at either end. Either entity can write a stream of bytes to the pipeline or read data bytes from the pipeline. However, because ADSP, like all other higher-level AppleTalk protocols, is a client of DDP, the data is actually sent as datagrams. This allows ADSP to correct transmission errors in a way that would not be possible for a true data stream connection. Thus, ADSP retains many of the advantages of a connectionless protocol while providing to its clients a connection-oriented full-duplex data stream.

An application that uses ADSP can treat the data to be transferred as continuous streams of data, or it can treat it as discrete messages to be interpreted individually. Applications that might use ADSP include server software applications such as mail servers, terminal emulation programs, or any application that requires two-way communication between computers. ADSP also includes features that let you authenticate the identity of the party at the other end of the connection and send encrypted data across the session, which is then decrypted at the other end. The authentication and encryption features of ADSP are referred to as *AppleTalk Secure Data Stream Protocol (ASDSP).*

AppleTalk Session Protocol

You can use the AppleTalk Session Protocol (ASP) to implement workstation applications that require an asymmetrical dialog with a server in which the workstation application initiates and controls the dialog. The workstation application tells the server application what to do and the server responds. ASP provides for the setting up, maintaining, and closing down of a session between a workstation and a server.

A workstation application that requires a state-dependent service should use ASP instead of ATP. ***State dependence*** means that the response to a request is dependent on a previous request. Consider the example of a workstation application connecting to a file server to read a file: before the application can read the file, it must have first issued a request to open the file. (For example, the AppleTalk Filing Protocol [AFP] uses ASP. However, only the client side of ASP is implemented on the Macintosh.) When a dialog is state dependent, all requests must be delivered in order and duplicate packets must not be sent: ASP provides for this.

An ATP transaction-based request, such as a workstation application requesting a server to return the time of day, is independent of other requests and not state dependent.

ASP assigns each session a unique identifier called a ***session reference number*** that allows more than one workstation to establish a session with the same server at the same time. For example, a server might use session reference numbers to distinguish between commands received from various clients of sessions.

ASP ensures that commands from a workstation are delivered without duplication and in the same order in which they were sent. ASP conveys the results of these commands back to the workstation. As long as the session is open, the workstation can request directory information, change filenames, and so forth. The file server must respond to the workstation's commands and cannot initiate any actions on its own.

Performing a Transaction

If you want to write an application that performs a transaction, you can use the AppleTalk Transaction Protocol (ATP). A transaction is an interaction between two applications that are clients of ATP in which one application, known as the *requester*, sends a request to the other application, known as the *responder*, to perform a task and return a response that reports the outcome of the task. The transaction request must fit in a single packet; however, the response can contain up to eight packets. ATP transactions are an efficient means of transporting small amounts of data across the network. ATP provides a reliable loss-free transport service. ATP's means of ensuring reliable delivery of data is based on the request-response paradigm as opposed to the data stream model that ADSP uses for reliable delivery of data.

You should use ATP

- if you want to send a small amount of data
- if your application requires delivery of all packets
- if your application can tolerate a minor degree of performance degradation
- if you do not want to incur the overhead and more extensive performance degradation involved in maintaining a session

ATP is useful for collecting status information; for example, a network management application might include a responder program on each node to which the central application sends out ATP requests asking for version information, such as the version of AppleTalk that the node is running. The responder program could check the version and send the information back to the main application in response to the request. Games that are based on request-and-response types of dialogs can make efficient use of ATP.

Sending and Receiving Data as Discrete Packets

Your application can use the Datagram Delivery Protocol (DDP) to transmit data in the form of packets across an AppleTalk internet. Because DDP provides best-effort delivery of datagrams with no recovery when packets are lost or discarded because of errors, it involves less overhead and provides for faster performance than do the higher-level protocols that add reliable delivery.

For applications, such as some games that don't require reliable delivery of data and can tolerate possible packet loss or diagnostic tools that retransmit at regular intervals to estimate averages, DDP suffices, and it offers the value of good performance. In fact, if you develop a game application that limits players to nodes on a single network, DDP will use short addressing headers on packets, requiring 8 fewer bytes per packet, which are faster to send.

If you are a network software developer who wants to develop a session-oriented protocol, a client-server protocol, or a transaction-based protocol that offers services different from those provided by ADSP, ASP, or ATP, you can design and implement your protocol as a client of DDP. However, this can entail providing your own server implementation in some cases. For a detailed description of DDP and the other AppleTalk protocols, see *Inside AppleTalk*, second edition.

If you use the DDP interface, you must provide a process called a *socket listener* to receive datagrams addressed to the socket. The chapter "Datagram Delivery Protocol (DDP)" in this book describes how to write a socket listener.

Measuring Packet-Delivery Performance

You can use the AppleTalk Echo Protocol (AEP) to measure the timing of send-receive cycles and to determine if another node is online. There is no application programming interface to AEP. However, to measure the round-trip packet delivery time from your node to another node, your application or process can send a packet that is addressed to the AEP socket, referred to as the *AEP Echoer*, on the destination node, and AEP will return a copy of that packet directly to you.

You can use this echo test as part of a diagnostic tool application, for example. A diagnostic tool could troubleshoot a suspect node and report how long it took the packet to travel to and from the node. Your application could use repeated transmissions to determine if a packet takes longer than the typical amount of time to reach the node, if it contains corrupted data, or if it doesn't make it back at all.

To determine if another node is on the network, you can send a packet to that node's AEP socket. For a conclusive test, you should send more than one packet, in case the first packet is lost or discarded by DDP.

Accessing AppleShare and Other File Servers

The AppleTalk Filing Protocol (AFP) provides an interface between an application and an AFP file server. For example, it allows workstations on an AppleTalk network to access files on AppleShare file servers. AFP uses the services of ASP.

Only the workstation side of AFP is implemented on the Macintosh. Few application developers use AFP because the existing File Manager commands perform most functions needed to access and manipulate files on an AppleShare server.

If you choose to use AFP, your application can provide support that allows a workstation user to use the workstation's own local or native file system commands to manipulate files on a remote node. The chapter "AppleTalk Filing Protocol (AFP)" in this book describes how to use AFP.

Receiving Packets Using a Virtual Node and Processing Them in a Custom Manner

Your application can use the AppleTalk multinode architecture to acquire node IDs that are in addition to the standard user node ID assigned to the system. You can use these virtual node IDs, called *multinodes,* to receive all broadcast packets and all AppleTalk packets addressed to the multinode. You can then process the packets in a custom manner. A multinode ID is not connected to the AppleTalk protocol stack above the data-link layer; this means that an application that uses a multinode is not connected to the AppleTalk protocol above the data-link level, and it cannot use their services. For example, Apple Remote Access (ARA) uses this multinode capability to implement remote access. The chapter "Multinode Architecture" describes how to acquire a multinode ID and send and receive packets using the multinodes.

The LAP Manager

The LAP Manager acts as an interface between the link types and the higher-level AppleTalk protocols. The LAP Manager contains a protocol handler that it attaches directly to the hardware device driver to receive 802.2 Type 1 packets for Ethernet, token ring, and FDDI. If your application handles 802.2 Type 1 packets, you must provide a protocol handler to read the packets and install your protocol handler as a client of the LAP Manager. A ***protocol handler*** is a piece of assembly-language code that controls the reception of a packet of a particular protocol type. When an 802.2 packet for your application arrives, the LAP Manager will call your protocol handler to read the packet.

The LAP Manager also provides and maintains a service called the ***AppleTalk Transition Queue (ATQ)*** that you can use to ensure that your application is not adversely affected when an AppleTalk transition occurs.

An example of an AppleTalk transition is an AppleTalk driver being closed or opened by another routine or the operating system. At any given time, there might be two or more applications running that use AppleTalk. If one of these applications closes the AppleTalk drivers, all AppleTalk applications are affected.

Your application can register itself with the AppleTalk Transition Queue by placing an entry in the queue. The LAP Manager sends a message to each entry in the AppleTalk Transition Queue when a transition occurs. Your application or other routines can also define their own AppleTalk events and call the AppleTalk Transition Queue to inform it that such an event occurred.

The AppleTalk Transition Queue also allows an application that uses the ***Flagship Naming Service*** to place an entry in the queue that enables it to stay informed as to changes to the flagship name. A ***flagship name*** is a personalized name that users can enter to identify their nodes when they are connected to an AppleTalk network. The flagship name is different from the Chooser name that a node uses for server-connection identification. The LAP Manager uses the transition queue message system to communicate name changes between applications and processes whenever the user resets the flagship name.

The chapter "Link-Access Protocol (LAP) Manager" in this book describes the LAP Manager services and interface. For more information about the LAP Manager, see the *Macintosh AppleTalk Connection Programmer's Guide*.

Using AppleTalk's link independence to write portable applications

If you write an application that uses one of the high-level AppleTalk protocols, such as ADSP or ATP, your program will run over any link type. A user running your application can switch between link types, for example, move from one type of network, such as token ring, to another, such as Ethernet, without affecting your program. The LAP Manager handles the interface and connection to the correct link-access protocol based on the link type the user selects. ◆

Directly Accessing a Driver for a Network Type

The .ENET, the .TOKN, or the .FDDI driver is normally called by the AppleTalk Manager through the AppleTalk connection file for the link type (EtherTalk, TokenTalk, or FDDITalk) when the user has selected one of these network types from the Network control panel. You can write your own protocol stack or application that uses one of these drivers directly rather than through AppleTalk.

The interface at this level allows you to open the driver and send data to it directly for transmission over the network. However, to receive data from the network, you need to provide a protocol handler written in assembly language.

For Phase 1 Ethernet packets, that is, the original version of Ethernet packets, you can read data directly from an Ethernet driver using the default protocol handler that Apple provides or your own protocol handler.

For IEEE 802.2 packets, you must use the interface to the Link-Access Protocol (LAP) Manager to attach your protocol handler to read data from an Ethernet, token ring, or FDDI driver. Token ring and FDDI support only 802.2 packets.

The chapter "Ethernet, Token Ring, and Fiber Distributed Data Interface" in this book describes how to use the interface for Phase 1 Ethernet packets. The chapter "Link-Access Protocol (LAP) Manager" in this book describes how to use the interface for IEEE 802.2 packets.

The AppleTalk Pascal Interface

This section provides an overview of the two execution modes that you can use to execute routines that belong to the AppleTalk protocol interfaces.

When your application calls an AppleTalk routine, you set a Boolean value as a parameter to the routine that directs the system software to execute the routine synchronously or asynchronously:

- If you set the routine to run synchronously, your application program cannot continue executing until the operation completes.
- If you set the routine to execute asynchronously, the system software returns control to your application program immediately and one of two methods is used to signal your program later when the operation completes; these methods are the use of a completion routine or a polling strategy.

The first version of the AppleTalk Pascal interfaces is now referred to collectively as the ***alternate interface.*** Routines belonging to the alternate interface that were executed asynchronously signaled the application that the operation had completed through the use of a network event.

Note
The use of network events introduced problems that were remedied by the creation of a new interface whose routines relied on the use of a completion routine or a result-field polling strategy rather than a network event as a completion-signaling mechanism for asynchronous calls. ◆

The new interface was designed to be similar to that of the Device Manager and the File Manager. Its routines use parameter blocks to pass input and output values. The interface glue code converts the parameter block values into a Device Manager `PBControl` call to the appropriate AppleTalk device driver. Called the ***preferred interface*** in the past, this interface is now the standard AppleTalk interface.

When writing new applications that use AppleTalk, you should use the routines belonging to the interface described in this book. Use of the alternate interface calls could cause compatibility problems with current and future system software, although the alternate interface is still provided in the header files for backward compatibility.

Note
For functions that execute asynchronously, you must not move or dispose of the parameter block before the function completes execution; while the function is executing, AppleTalk owns the memory that you allocated for the function's use. *After* the call returns, you need to dispose of the memory allocated for the parameter block unless you intend to reuse the parameter block, for example, for another function. ◆

Executing Routines Synchronously or Asynchronously

Your program can execute the routines that make up the interface to the AppleTalk protocols either synchronously or asynchronously. ***Synchronous execution*** means that your program is prevented from doing any other processing until the current operation completes. ***Asynchronous execution*** means that the system returns control to your program after your program calls the routine so that your program can continue with other processing while the asynchronous operation completes.

If you execute a routine synchronously, the call does not return until the operation completes; you do not have to use a completion routine that runs at interrupt level or poll a result field to determine when the operation completes; on the other hand, your program cannot continue running until the call returns, which causes the system to come to a standstill. Synchronous calls are useful for operations that execute and return to the calling program quickly, such as opening or closing sockets. On an AppleTalk internet, data is transferred between sockets, which must be opened before they can be used and closed when they are no longer needed.

Calling a routine asynchronously directs the system software to begin the operation process now, return control to the calling program, then complete execution of the routine as soon as possible. Asynchronous execution eliminates program execution delay time, but it requires that your application provide a means of determining when the operation has completed execution. There are two methods an application can use to determine when an operation completes execution:

- An application can provide a completion routine to be called at interrupt level.
- An application can poll the routine's parameter block result field.

The parameter block that is used to contain input and output information for a function includes a result field called `ioResult`. When your application calls a function asynchronously,

- the driver executes the function, if possible.
- if the driver is busy, the driver queues the function and sets the `ioResult` field to 1.

When the function completes execution, the driver sets the result field to a value that indicates either that no error occurred (`noErr`) or an error condition code value that identifies the type of error.

Polling the Result Field

Your application can poll the result field to determine when the result value changes. Your application can use the polling process to inform the user that the system is still busy performing the operation that handles the request; for processes that may take a long time, your application can display a progress dialog box to the user.

Note
If you use polling, you must set the call's parameter block `ioCompletion` field to `NIL`. ◆

Using a Completion Routine

Instead of polling the result field, your application can supply a completion routine to be executed at interrupt level when the operation completes. You provide the address of the completion routine in the call's parameter block `ioCompletion` field. Because completion routines are executed at interrupt level, they cannot call any routines that move memory.