This chapter describes the AppleTalk functions and services that do not belong to a specific AppleTalk protocol interface but that apply to AppleTalk as a whole.

The chapter describes how to

- obtain a wide variety of information about AppleTalk and the network environment of your node, including the maximum number of protocol handlers and concurrent NBP calls that the installed .MPP driver supports
- obtain the addresses of your node and its local internet router
- enable intranode delivery, which lets you send packets to your own application or other applications and processes running on the same node as yours
- determine if the AppleTalk Phase 2 drivers are installed on your system
- select a node ID in the server range
- open the .MPP and .XPP drivers

The .MPP driver opens the .ATP driver. The chapter "AppleTalk Data Stream Protocol (ADSP)" in this book describes how to open the .DSP driver. Although Apple Computer, Inc. recommends that you not close any of the AppleTalk drivers because other applications that are coresident may be using them, this chapter explains how to close the .MPP driver, if, for some reason, you must.

About the AppleTalk Utilities

The AppleTalk Utilities are a group of diverse functions, some of which allow you to obtain information about AppleTalk and the networking environment of your node and some of which allow you change values that affect AppleTalk features.

The `PGetAppleTalkInfo` function returns a wide range of information, including some information that other functions belonging to the AppleTalk Utilities also return. For example, both `PGetAppleTalkInfo` and `GetNodeAddress` return the node ID and network address of the user node that is running your application. The `PGetAppleTalkInfo` function returns the node ID and the network number of the last router from which the node that is running your application has heard; the `GetBridgeAddress` function also returns the node ID of the internet router on your node's local network.

Note
The `PGetAppleTalkInfo` function was developed and made available after the `GetNodeAddress` and `GetBridgeAddress` functions. Apple Computer, Inc. recommends that you use the `PGetAppleTalkInfo` function to obtain addressing information for a user node or router instead of using the `GetNodeAddress` and `GetBridgeAddress` functions. ◆

Although the AppleTalk interface does not include a function that you can use to direct AppleTalk to select a node ID from the server node range when you open AppleTalk, this chapter describes how you can do this. If your application or the application that

opened AppleTalk directed AppleTalk to assign a server node ID to the node, the `PGetAppleTalkInfo` function will return a flag that tells you this request was made.

AppleTalk includes a feature called ***intranode delivery*** that allows two programs running on the same node to communicate with each other through the AppleTalk protocols. The AppleTalk Utilities include the `PSetSelfSend` function, which you can use to enable or disable intranode delivery. The `PGetAppleTalkInfo` function will tell you if intranode delivery is on or off.

Using the AppleTalk Utilities

This section describes how to use some of the functions and services that make up the AppleTalk Utilities. It explains how to

- check the version of the AppleTalk drivers that are installed
- get information about the .MPP driver and the network environment
- get the address of your node and locate your local router
- enable intranode delivery
- request AppleTalk to assign to your node an ID that is in the range of numbers that are reserved for server nodes

Determining Whether AppleTalk Phase 2 Drivers Are Supported

Once the .MPP driver has been loaded into memory, you can use the `Gestalt` function with the `gestaltAppleTalkVersion` selector to check the version of AppleTalk. The `Gestalt` function returns the version of the .MPP driver. If the version is equal to or greater than 53, then the .MPP driver supports AppleTalk Phase 2.

Alternatively, you can call the `SysEnvirons` function. If the `atDrvrVersNum` field of the `SysEnvRec` data structure returned by this function is equal to or greater than 53, then the .MPP driver supports AppleTalk Phase 2.

Getting Information About the .MPP Driver and the Network Environment

This section describes how you can use the `PGetAppleTalkInfo` function to obtain information about the installed version of the .MPP driver, the network environment, and the .MPP driver's maximum capacities, such as the number of sockets and the number of NBP calls that the .MPP driver supports. The .MPP driver implements these protocols:

- Datagram Delivery Protocol (DDP)
- Routing Table Maintenance Protocol (RTMP) stub

- Name-Binding Protocol (NBP)
- AppleTalk Echo Protocol (AEP)

Before you call the `PGetAppleTalkInfo` function, you must allocate memory for and define a parameter block of type `MPPParmType`. The section "MPP Parameter Block" beginning on page 2-9 shows this data structure. You must also allocate memory for and provide pointers to the data buffers into which the `PGetAppleTalkInfo` function returns the data-link address and zone name for extended networks.

The `PGetAppleTalkInfo` function's Boolean parameter allows you to specify whether the function is to be executed synchronously or asynchronously. This function is generally executed synchronously. (For information on these two modes, see the chapter "Introduction to AppleTalk" in this book.)

The `PGetAppleTalkInfo` function returns the following information:

- a pointer to the MPP global variables
- a pointer to the .MPP driver's device control entry (DCE) data structure
- configuration flags that indicate the status of certain conditions that are set at startup
- a value (the `selfSend` flag) that indicates whether the node can send packets to itself (See "Sending Packets to Applications and Processes on Your Own Node" on page 2-6 and "Enabling Intranode Delivery of DDP Packets" on page 2-15 for more information.)
- the range of network numbers for the network to which the node is attached
- the 8-bit node ID and 16-bit network number of the node
- the 8-bit node ID and 16-bit network number of the last router from which the node has heard
- the maximum capacities of the .MPP driver, such as the maximum number of protocol handlers and the maximum number of static sockets allowed by this driver
- a pointer to the registered names queue
- the address of the node on the underlying data link (for example, the Ethernet hardware address)
- the node's zone name

The data-link address and the zone name are returned only for extended networks—that is, network types that allow more than one network number per network. You use the `laLength` parameter to specify the length of the data-link address you want returned; the function returns the actual length of the data in the `laLength` parameter and returns the data in the buffer you provide.

The `ExtendedBit` flag returned by the `PGetAppleTalkInfo` function is `TRUE` if the node is connected to an extended AppleTalk network. (The `ExtendedBit` flag is bit 15 of the configuration parameter returned by this function.) Note that the presence of the AppleTalk Phase 2 drivers does not of itself indicate that the node is connected to an extended network. For more information, see "PGetAppleTalkInfo" beginning on page 2-11.

Note
Always use the `PGetAppleTalkInfo` function to obtain information about the .MPP driver. You cannot rely on the validity of the MPP global variables pointed to by the `varsPtr` parameter block field value for this information. ◆

Getting the Address of Your Node or Your Local Router

You can use the AppleTalk Utilities `GetNodeAddress` function to get the node ID of the node that is running your application and the number of the network to which that node is connected.

Note
If `GetNodeAddress` returns a network number of 0, this means that there is no internet router available. However, your application or process should call `GetBridgeAddress` to determine if there are router-like services, such as Apple Remote Access (ARA), available to that node. ◆

To locate your local router, you can first call `GetNodeAddress` for the router's network number; the network number that `GetNodeAddress` returns for a node is also valid for the internet router on that local network. To get the node ID part of a local router's address, you can call the `GetBridgeAddress` function. If there is not a router on the local network, `GetBridgeAddress` returns a function result of 0.

Note
You can also use `GetZoneList` to determine if there is a router on the local network. For information on `GetZoneList`, see the chapter "Zone Information Protocol (ZIP)" in this book. ◆

Sending Packets to Applications and Processes on Your Own Node

Because more than one application or process can be running on a single node at the same time, it is reasonable to assume that you may want to send packets from your application or process to other applications and processes running on the same node. To support this, AppleTalk includes a function that lets you turn on (or off) an intranode delivery feature.

When intranode delivery is on, two programs running on the same node can communicate with each other through the AppleTalk protocols. You can address and send a packet to another application or process that is an internet socket client running on your own node from any of the AppleTalk protocols that provide programming interfaces.

You use the `PSetSelfSend` function to enable or disable intranode delivery. The `PSetSelfSend` function returns the value of the previous setting, so that you can save it and reinstate the value later if it differs from the setting that you specify. For more information about enabling or disabling intranode delivery, see "PSetSelfSend" beginning on page 2-15.

Note
Intranode delivery applies to user node applications and processes. Sending packets between a multinode application and user node applications on the same machine is independent of the intranode delivery feature. A multinode is treated as a virtual node distinct from the user node; both the user node and the multinode have their own node IDs. ◆

Selecting a Node in the Server Range

AppleTalk node IDs are divided into two classes: ***user node IDs*** and ***server node IDs.***

- User node IDs are in the range 1–127 ($01–$7F).
- Server node IDs are in the range 128–254 ($80–$FE).

AppleTalk's dynamic node assignment occurs through a process in which the node acquiring a node ID sends out enquiry packets to determine if the ID that the node suggests is available. Although unlikely, problems can occur if a node that owns the suggested ID fails to respond to the enquiry because it is busy.

User nodes are switched on and off more frequently than are server nodes. Separating user node ID assignment from server node ID assignment allows for different degrees of verification.

Within the user node ID range, verification is performed quickly with fewer retransmissions of the enquiry control packet than are sent for server node ID verification; this decreases the initialization time for user nodes. A more thorough node ID verification is performed for servers. This scheme increases the initialization time for server nodes but is not detrimental to the server's operation because server nodes are rarely switched on and off.

You can start up AppleTalk so that it will assign a node ID within the server range by making an extended `Open` call to the .MPP driver. To do this, you set the immediate bit in the `_Open` trap. To request a server node ID, set to 1 the high bit (bit 31) of the extension longword field `ioMix` in the extended call. Set to 0 the remaining bits in the `ioMix` field and the bits of all the other unused fields in the queue element. The code in Listing 2-1 sets the high bit in the `ioMix` field, then it calls an assembly-language routine that is not shown in this listing, `PBOpenImmedSync`, to make the extended open call. The code uses the following global constants:

```
SPConfig       = $01FB;
portBClearMask = $F0;
```

The code in Listing 2-1 assumes that the .MPP driver is not currently open. It is important to remember that you can only request a server node ID when you first open the .MPP driver.

Listing 2-1 Opening the .MPP driver and obtaining a node ID in the server range

```
FUNCTION PBOpenImmedSync(paramBlock: ParmBlkPtr): OSErr;
INLINE $205F,$A200,$3E80;
FUNCTION OpenNodeInServerRange: OSerr;
IMPLEMENTATION
FUNCTION OpenNodeInServerRange: OSerr;
VAR
   MPPPtr:        ParmBlkPtr;
   err:           OSerr;
   MPPName:       Str31;
   SpConfigPtr:   Ptr;
BEGIN
   IF IsMPPOpen THEN
      BEGIN
         OpenNodeInServerRange := openErr;
      END
   ELSE
      BEGIN
         SPConfigPtr  := Ptr(SPConfig);
         SPConfigPtr^ := BYTE(BAND(SPConfigPtr^, portBClearMask));
         SPConfigPtr^ := BYTE(BOR(SPConfigPtr^, UseATalk));
         MPPName := '.MPP';
         MPPPtr := ParmBlkPtr(NewPtrClear(sizeof(ParamBlockRec)));
         MPPPtr^.ioMix := Ptr($80000000);
         MPPPtr^.ioNamePtr := @MPPName;
         OpenNodeInServerRange := PBOpenImmedSync(MPPPtr);
      END
END;
```

AppleTalk Utilities Reference

This section describes the data structure and the routines that make up the AppleTalk Utilities. The "Data Structures" section shows the MPP parameter block required for the `PSetSelfSend` and the `PGetAppleTalkInfo` functions.

The "Routines" section describes the routines for

- getting information about the installed .MPP driver and the current network environment
- enabling intranode delivery
- getting the addresses of your node and your local internet router
- opening the .MPP and .XPP drivers (The .MPP driver opens the .ATP driver.)

Data Structures

This section describes the MPP parameter block that you use for the `PSetSelfSend` and `PGetAppleTalkInfo` functions.

MPP Parameter Block

The `PSetSelfSend` and `PGetAppleTalkInfo` functions require a pointer to the MPP parameter block. The `MPPParamBlock` data type defines the MPP parameter block.

- The `PGetAppleTalkInfo` function uses the MPP parameter block with the `GetAppleTalkInfoParm` variant record to pass information to and receive it from the .MPP driver.
- The `PSetSelfSend` function uses the MPP parameter block with the `SetSelfSendParm` variant record to pass information to and receive it from the .MPP driver. The `MPPParamBlock` data type defines the MPP parameter block.

This section defines the fields common to both of these functions. The fields for the variant records are defined in the function description that uses the record.

```
TYPE
   MPPParmType    =     (...SetSelfSendParm,
                         GetAppleTalkInfoParm...);
   MPPPBPtr       =     ^MPPParamBlock;
   MPPParamBlock  =
   PACKED RECORD
      qLink:              QElemPtr;      {reserved}
      qType:              Integer;       {reserved}
      ioTrap:             Integer;       {reserved}
      ioCmdAddr:          Ptr;           {reserved}
      ioCompletion:       ProcPtr;       {completion routine}
      ioResult:           OSErr;         {result code}
      ioNamePtr:          StringPtr;     {reserved}
      ioVRefNum:          Integer;       {reserved}
      ioRefNum:           Integer;       {driver reference }
                                         { number}
      csCode:             Integer;       {primary command code}
   CASE MPPParmType OF
      SetSelfSendParm:
         (newSelfFlag:    Byte;          {self-send toggle flag}
          oldSelfFlag:    Byte);         {previous self-send }
                                         { state}
      GetAppleTalkInfoParm:
         (version:        Integer;       {requested info version}
         varsPtr:         Ptr;           {pointer to MPP }
                                         { variables}
```

```
        DCEPtr:          Ptr;          {pointer to MPP DCE}
        portID:          Integer;      {port number [0..7]}
        configuration:   LongInt;      {32-bit configuration }
                                       { word}
        selfSend:        Integer;      {nonzero if self-send }
                                       { enabled}
        netLo:           Integer;      {low value of network }
                                       { range}
        netHi:           Integer;      {high value of network }
                                       { range}
        ourAddr:         LongInt;      {our 24-bit AppleTalk }
                                       { address}
        routerAddr:      LongInt;      {24-bit address of }
                                       { last router}
        numOfPHs:        Integer;      {max. number of }
                                       { protocol handlers}
        numOfSkts:       Integer;      {max. number of static }
                                       { sockets}
        numNBPEs:        Integer;      {max. concurrent NBP }
                                       { requests}
        ntQueue:         Ptr;          {pointer to registered }
                                       { name queue}
        LAlength:        Integer;      {length in bytes of }
                                       { data-link address}
        linkAddr:        Ptr;          {data-link address }
                                       { returned}
        zoneName:        Ptr);         {zone name returned}
END;
```

Field descriptions

`ioCompletion` A pointer to a completion routine that you can provide. When you execute the `PGetAppleTalkInfo` function or the `PSetSelfSend` function asynchronously, the .MPP driver calls your completion routine when it completes execution of the function. Specify `NIL` for this field if you do not wish to provide a completion routine. If you execute the function synchronously, the .MPP driver ignores the `ioCompletion` field.

`ioResult` The result of the function. When you execute the function asynchronously, the function sets this field to 1 and returns a function result of `noErr` as soon as the function begins execution. When the function completes execution, it sets the `ioResult` field to the actual result code.

`ioRefNum` The .MPP driver reference number. The MPW interface fills in this field.

`csCode`	The routine selector command code of the .MPP command to be executed. The MPW interface fills in this field. For the `PGetAppleTalkInfo` function, `csCode` is always `GetATalkInfo`. For the `PSetSelfSend` function, `csCode` is always `setSelfSend`.

Routines

This section describes the routines that you use to obtain information about AppleTalk and the network environment, enable intranode delivery of DDP packets, obtain your node's address and your local network router's address, and open and close the .MPP, .ATP, and .XPP drivers.

Obtaining Information About the .MPP Driver and the Current Network Environment

You can use the `PGetAppleTalkInfo` function to obtain a wide variety of information about the .MPP driver that is installed on the node that is running your application and the network environment of that node. Among the information that the `PGetAppleTalkInfo` function returns are

- the address and zone name of the node that is running your application
- the number of concurrent NBP calls that the installed .MPP driver supports
- the range of network numbers for the network, if it is an extended network

PGetAppleTalkInfo

The `PGetAppleTalkInfo` function returns information about the currently installed version of the .MPP driver and the network environment.

```
FUNCTION PGetAppleTalkInfo (thePBptr: MPPPBPtr; async:
                                Boolean): OSErr;
```

`thePBptr`	A pointer to an MPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSerr`	The result code.
→	`ioRefNum`	`Integer`	The .MPP driver reference number.
→	`csCode`	`Integer`	Always `GetATalkInfo`.
→	`version`	`Integer`	The version of the function.
←	`varsPtr`	`Ptr`	A pointer to the MPP globals.

continued

←	DCEPtr	Ptr	A pointer to DCE for the .MPP driver.
←	portID	Integer	The port number.
←	configuration	LongInt	The configuration flags.
←	selfSend	Integer	Nonzero if self-sending is enabled.
←	netLo	Integer	The low value of the network range.
←	netHi	Integer	The high value of the network range.
←	ourAddr	LongInt	The local 24-bit AppleTalk address.
←	routerAddr	LongInt	The 24-bit address of the router.
←	numOfPHs	Integer	The maximum number of protocol handlers.
←	numOfSkts	Integer	The maximum number of static sockets.
←	numNBPEs	Integer	The maximum concurrent NBP requests.
←	ntQueue	Ptr	A pointer to registered names table.
↔	LAlength	Integer	The length in bytes of data-link address (extended networks only).
→	linkAddr	Ptr	A pointer to data-link address buffer (extended networks only).
→	zoneName	Ptr	A pointer to zone name buffer.

Field descriptions

`version` The version number of the `PGetAppleTalkInfo` function you are calling. For version number 53 and greater of the .MPP driver, this number is always 1.

`varsPtr` A pointer to the MPP global variables. This parameter is reserved for the use of Apple Computer, Inc.; you cannot rely on the validity of the variables pointed to by this parameter.

`DCEPtr` A pointer to the device control entry (DCE) data structure for the .MPP driver. For information about the DCE, see the chapter "Device Manager" in *Inside Macintosh: Devices*.

`portID` The port number for the .MPP driver. The port number is always 0 unless you are requesting information for an .MPP driver being used by a router.

`configuration` A 32-bit longword of configuration flags. The following flags are currently defined:

Bit	Flag	Description
31	`SrvAdrBit`	`TRUE` (equal to 1) if the routine that opened the .MPP driver requested a server node number. For more information on server nodes, see "Selecting a Node in the Server Range" on page 2-7. This flag indicates only that the server node number was requested, not that it was returned. Some AppleTalk data links, such as EtherTalk, TokenTalk, and FDDITalk, do not honor a request for a server node number.

continued

Bit	Flag	Description
30	RouterBit	TRUE (equal to 1) if an AppleTalk internet router was loaded at system startup (that is, there's a router operating on the same node as your application). A router can be loaded and not active.
15	ExtendedBit	TRUE (equal to 1) if the node is on an extended network. Testing this bit is the only way to determine whether you are on an extended network.
7	BadZoneHintBit	TRUE (equal to 1) if the zone name of the node you are on was not the same as the zone name stored in parameter RAM (sometimes referred to as the *zone name hint*) when the .MPP driver was opened. If the zone name hint is invalid, then the AppleTalk Manager uses the default zone for the network. The default zone is defined by the network administrator.
6	OneZoneBit	TRUE (equal to 1) if only one zone is assigned to your extended network or if you are not on an extended network. Use the ExtendedBit flag to determine whether you are on an extended network.

`selfSend` The ability of a node to send packets to itself. This feature, called *intranode delivery,* is enabled when this parameter is nonzero. Use the `PSetSelfSend` function, which is described beginning on page 2-15, to enable or disable this feature.

`netLo` The low value of the range of network numbers on the local cable. Only extended networks can have a range of network numbers. For a nonextended network, this parameter returns the network number.

`netHi` The high value of the range of network numbers on the local cable. Only extended networks can have a range of network numbers. For a nonextended network, this parameter returns the network number.

`ourAddr` The 24-bit AppleTalk network address of the node you are on. The least significant byte of the longword is the node ID. The middle 16 bits are the network number. The most significant byte of the longword is reserved for use by Apple Computer, Inc.

`routerAddr` The 24-bit AppleTalk network address of the last router from which your node heard traffic. The least significant byte of the longword is the node ID. The middle 16 bits are the network number. The most significant byte of the longword is reserved for use by Apple Computer, Inc. You should always use this address when you want to communicate with a router.

`numOfPHs`	The maximum number of protocol handlers that this .MPP driver allows.
`numOfSkts`	The maximum number of statically assigned sockets that this .MPP driver allows. Statically assigned sockets are described in *Inside AppleTalk,* second edition. For more information about sockets, see the chapter "Datagram Delivery Protocol (DDP)" in this book.
`numNBPEs`	The maximum number of concurrent requests to NBP that this .MPP driver allows.
`ntQueue`	A pointer to the first entry in the names table for the local node. You can use NBP routines to look up and register names in the names table. The names table is described in the chapter "Name-Binding Protocol (NBP)" in this book.
`LAlength`	The number of bytes of the data-link address that the function should place in the buffer pointed to by the `LinkAddr` parameter. You use this parameter when you call the `PGetAppleTalkInfo` function on a node on an extended network. If you request more bytes than the total number of bytes in the address, then the function returns in the `LAlength` parameter the actual number of bytes it placed in the buffer. If the address is longer than the size of the buffer, then the `PGetAppleTalkInfo` function fills the buffer and returns in the `LAlength` parameter the actual length of the address, not the number of bytes returned. The function does *not* return an error when the buffer is too large or too small for the address. A value of 6 bytes for `LAlength` is sufficient for most purposes.
`linkAddr`	A pointer to a buffer for the data-link address returned for extended networks only. You use the `LAlength` parameter to specify the number of bytes of the address that you want placed in this buffer. You must allocate a buffer large enough to hold the number of bytes you specify. Specify `NIL` for this parameter if you do not want the function to provide a data-link address.
`zoneName`	A pointer to a buffer into which the `PGetAppleTalkInfo` function places the local node's zone name. You must allocate a buffer of at least 33 bytes to hold this data, or you must specify `NIL` for the `zoneName` parameter if you do not want to obtain the zone name. This field is returned only if the node is on an extended network.

DESCRIPTION

The `PGetAppleTalkInfo` function returns a variety of information about the current networking environment. For example, it returns information telling you whether or not applications running on the node can send packets to themselves or to other applications or processes on the same node. An application can call `PGetAppleTalkInfo` to determine if the node on which it is running has an ID that falls within the server node ID range. It can also obtain the address of the last router that the node communicated with and the node's own address.

You must allocate memory for and define a parameter block of type `MPPParmType` and pass that parameter block's pointer to `PGetAppleTalkInfo` when you call the function. You must also allocate memory for and provide pointers to the data buffers into which

the `PGetAppleTalkInfo` function returns the data-link address and zone name. You pass a pointer to the buffer for the returned data-link address as the value of the `linkAddr` field. You pass a pointer to the buffer for the returned zone name as the value of the `zoneName` parameter block field.

SPECIAL CONSIDERATIONS

If the node on which your application is running happens also to be running AppleTalk internet router software in the background, more than one set of MPP global variables may be in RAM. To make sure you obtain information about the .MPP driver that handles application software, always use the `PGetAppleTalkInfo` function rather than the Device Manager's `PBControl` function. However, if you want to use the `PBControl` function, you must use a device driver reference number of –10 for the .MPP driver.

The memory that you allocated for the parameter block and data buffers belongs to the .MPP driver until the `PGetAppleTalkInfo` function completes execution. The memory must be nonrelocatable. After the `PGetAppleTalkInfo` function completes execution, you can reuse the memory or release it.

ASSEMBLY-LANGUAGE INFORMATION

If you use assembly language to call this function, you must use a device driver reference number of –10 for the .MPP driver.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Version number is too high

Enabling Intranode Delivery of DDP Packets

This section describes how the `PSetSelfSend` function allows applications and processes running on the same node to send packets to one another.

PSetSelfSend

The `PSetSelfSend` function enables or disables the AppleTalk intranode delivery service.

```
FUNCTION PSetSelfSend (thePBptr: MPPPBPtr; async: Boolean): OSErr;
```

`thePBptr`	A pointer to an MPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .MPP driver reference number.
→	`csCode`	`Integer`	Always `setSelfSend`.
→	`newSelfFlag`	`Byte`	A flag that turns intranode delivery on or off.
←	`oldSelfFlag`	`Byte`	A flag that reports the previous state of intranode delivery, whether it was on or off.

Field descriptions

`newSelfFlag` A flag that enables or disables the intranode delivery feature. Set this field to a nonzero number to enable the feature; set it to zero to turn off the feature.

`oldSelfFlag` A flag indicating the previous state of the intranode delivery feature. The `PSetSelfSend` function returns this value. A nonzero value indicates that intranode delivery was enabled; a value of zero indicates it was disabled.

DESCRIPTION

The `PSetSelfSend` function turns on or off the intranode delivery feature that allows you to send a packet to another socket on the same node. You can use this feature, for example, to send data from an application to a print spooler that is running in the background on the same node.

When `PSetSelfSend` is enabled, you can send packets to socket clients on your node from all levels of the AppleTalk protocol stack for which there are programming interfaces. The `PSetSelfSend` function returns in the `oldSelfFlag` field the previous setting for the intranode delivery feature so that you can restore it later, if you want to. Because intranode delivery is enabled on most systems running AppleTalk, you should assume that it is turned on and take this into account when you write your code.

Note that intranode delivery applies to the user node applications. Sending packets between a multinode application and user node applications on the same machine is independent of the intranode delivery feature. A multinode is treated as a virtual node distinct from the user node; both the user node and the multinode have their own node IDs.

SPECIAL CONSIDERATIONS

Enabling or disabling the intranode delivery feature affects the entire node. For example, an application that uses NBP to look up names and then display them to a user might not expect to receive names of other network-visible entities within its own node; when intranode delivery is enabled, this will occur.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `PSetSelfSend` function from assembly language, call the `_Control` trap macro with a value of `setSelfSend` in the `csCode` field of the parameter block.

RESULT CODES

`noErr`	0	No error

Getting the Addresses of Your Node and Local Internet Router

This section describes the `GetNodeAddress` and `GetBridgeAddress` functions, which you can use to get the address of the node that is running your application or process and to determine if the local network to which that node is connected includes a router. If there is a router on the local network, `GetBridgeAddress` will return the node ID of that router. The router's network number is the same as that of your local network.

GetNodeAddress

The `GetNodeAddress` function returns the current node ID and network number of the node on which the calling program is running.

```
FUNCTION GetNodeAddress (VAR myNode,myNet: Integer): OSErr;
```

`myNode` The node ID of the node on which your application or process is running.

`myNet` The network number of the network to which the node is attached that is running your application or process. If `myNet` returns 0, this means that there is no internet router available. However, your application or process should call `GetBridgeAddress` to determine if there are router-like services available to that node.

DESCRIPTION

The `GetNodeAddress` function returns the address of a node on a network. If the network is not an extended network, the network number that `GetNodeAddress` returns is 0. Note that even if `GetNodeAddress` returns a network number of 0, there may be a router service on the local network. For example, a node can be on a network whose network number is 0 and be connected to a remote network through Apple Remote Access (ARA).

If the .MPP driver is not installed, the `GetNodeAddress` function returns a function result of `noMPPErr`. To install the .MPP driver, open it using the Device Manager's `OpenDriver` function or the `MPPOpen` function.

ASSEMBLY-LANGUAGE INFORMATION

This function is implemented in the MPW glue code only. It is not accessible from assembly language.

RESULT CODES

`noErr`	0	No error
`noMPPErr`	–3102	The .MPP driver is not installed

GetBridgeAddress

The `GetBridgeAddress` function returns the node ID of the router on your local network.

```
FUNCTION GetBridgeAddress: Integer;
```

DESCRIPTION

The `GetBridgeAddress` function returns the current node ID of an internet router in the low-order byte of the function result. If the function result is 0, there is no router on the local network. The router's network number is that of the local network; you can use the `GetNodeAddress` function to get the network number.

ASSEMBLY-LANGUAGE INFORMATION

This function is implemented in the MPW glue code only. It is not accessible from assembly language.

SEE ALSO

To obtain the network number of the local network, use the `GetNodeAddress` function described on page 2-17.

Opening and Closing Drivers

This section describes the functions that you can use to open the .MPP and .XPP drivers, `MPPOpen` and `OpenXPP`. The .MPP driver opens the .ATP driver. This section also describes the function that closes the .MPP driver, `MPPClose`.

The `MPPOpen` and `OpenXPP` functions are included to provide a complete description of the AppleTalk programmatic interface. Apple Computer, Inc. recommends that you use the Device Manager's `OpenDriver` function to open the .MPP and .XPP drivers. In addition to opening a driver, the `OpenDriver` function returns the driver reference number. If the driver is already open, the `OpenDriver` function simply returns the driver reference number. For information on the `OpenDriver` function, see the chapter "Device Manager" in *Inside Macintosh: Devices*.

The .MPP, .ATP, and .XPP drivers must always be open before you can use the AppleTalk protocols that they implement. The .MPP driver must be open before you open the .XPP driver. How to open the .DSP driver is described in the chapter "AppleTalk Data Stream Protocol (ADSP)" in this book.

▲ **WARNING**
Because coresident programs might also be using AppleTalk, you should not close the AppleTalk drivers. ▲

This section also includes the `IsMPPOpen` and `IsATPOpen` functions that determine if the .MPP and the .ATP drivers are already open.

MPPOpen

If the .MPP driver has not already been opened, the `MPPOpen` function opens the .MPP driver, initializes the driver's variables, and assigns a node ID to the Macintosh computer.

```
FUNCTION MPPOpen: OSErr;
```

DESCRIPTION

The `MPPOpen` function first determines whether the .MPP driver has already been opened. If it has, `MPPOpen` returns an error code. If the .MPP driver is not open, `MPPOpen` loads the driver into the system heap and then initializes the driver's variables before dynamically assigning a node ID to the system. It also loads the .ATP driver and the NBP code into the system heap.

Apple Computer, Inc. recommends that you use the Device Manager's `OpenDriver` function to open the .MPP driver instead of using the `MPPOpen` function.

SPECIAL CONSIDERATIONS

For versions of AppleTalk before AppleTalk version 56, if serial port B isn't configured for AppleTalk or if it is already in use, the .MPP driver is not loaded and the `portInUse` result code is returned.

RESULT CODES

`noErr`	0	No error
`portInUse`	–97	Driver open error code indicating that the port is in use
`portNotCf`	–98	Driver open error code indicating that the parameter RAM is not configured for this connection

SEE ALSO

The `MPPOpen` function does not return the .MPP driver reference number, as the `OpenDriver` function does. For information on the `OpenDriver` function, see the chapter "Device Manager" in *Inside Macintosh: Devices*.

MPPClose

The `MPPClose` function closes the .MPP driver and removes from memory any data structures associated with it.

```
FUNCTION MPPClose: OSErr;
```

DESCRIPTION

In addition to closing the .MPP driver, the `MPPClose` function also closes and removes from memory the .ATP driver and the NBP code if they are installed. Calling `MPPClose` completely disables AppleTalk.

▲ **WARNING**
Apple Computer, Inc. strongly recommends that you *not* use this call because other coresident applications could also be using AppleTalk. ▲

Calling `MPPClose` completely disables AppleTalk.

SPECIAL CONSIDERATIONS

If the current connection is LocalTalk, `MPPClose` also returns the use of port B to the serial driver.

RESULT CODES

`noErr`	0	No error

IsMPPOpen

The `IsMPPOpen` function determines and reports whether or not the .MPP driver is loaded and running.

```
FUNCTION IsMPPOpen: Boolean;
```

DESCRIPTION

If the .MPP driver is open, the `IsMPPOpen` function returns a value of `TRUE`; if the .MPP driver is not open, it returns `FALSE`. If you want to obtain a node ID in the server range, you can request the assignment only when you first open the .MPP driver. In this case, you can use the `IsMPPOpen` function to determine if the .MPP driver has already been opened.

RESULT CODES

`noErr`	0	No error

SEE ALSO

You can also use the Device Manager's `OpenDriver` function to ensure that the .MPP driver is open. If it is not, `OpenDriver` will open the .MPP driver and return the driver reference number. If the .MPP driver is already open, the `OpenDriver` function will return the reference number without performing additional processing, and therefore without incurring much additional overhead.

IsATPOpen

The `IsATPOpen` function determines and reports whether or not the .ATP driver is loaded and running.

```
FUNCTION IsATPOpen: Boolean;
```

DESCRIPTION

If the .ATP driver is open, the `IsATPOpen` function returns a value of `TRUE`; if the .ATP driver is not open, it returns `FALSE`. Because the .MPP driver opens the .ATP driver, this function is seldom used. It is included to provide a complete description of the AppleTalk programmatic interface.

RESULT CODES

`noErr`	0	No error

SEE ALSO

To open the .ATP driver, you open the .MPP driver. You can use the Device Manager's `OpenDriver` function to ensure that the .MPP driver is open. If the .MPP driver is open, then the .ATP driver is also open. If the .MPP and .ATP drivers are already open, the `OpenDriver` function will return the .MPP driver reference number without performing additional processing, and therefore without incurring much additional overhead.

For information on the `OpenDriver` function, see the chapter "Device Manager" in *Inside Macintosh: Devices.*

OpenXPP

The `OpenXPP` function opens the .XPP driver and returns the driver reference number.

```
FUNCTION OpenXPP (VAR xppRefnum: Integer): OSErr;
```

`xppRefnum` The .XPP driver reference number, which the function returns.

DESCRIPTION

Before you can use the protocol interfaces (ZIP, ASP, and AFP) that are implemented by the .XPP driver, you must open the driver. You can use the `OpenXPP` function to open the .XPP driver, or you can call the Device Manager's `OpenDriver` function. In either case, before you open the .XPP driver, you must ensure that the .MPP driver and the .ATP driver are open.

Apple Computer, Inc. recommends that you use the Device Manager's `OpenDriver` function to open the .XPP driver instead of using the `OpenXPP` function. The `OpenXPP` function is included to provide a complete description of the AppleTalk programmatic interface.

SPECIAL CONSIDERATIONS

Under most circumstances, you should not close the .XPP driver because other applications and processes could be using it. However, if you must close the .XPP driver, you can use the Device Manager's `CloseDriver` function. The `CloseDriver` function should be used only by system-level applications.

RESULT CODES

`noErr`	0	No error
`portInUse`	-97	Either AppleTalk is not open or the AppleTalk port is in use by another driver

SEE ALSO

The `OpenXPP` function does not return the .MPP driver reference number, as does the `OpenDriver` function. For information on the `OpenDriver` and `CloseDriver` functions, see the chapter "Device Manager" in *Inside Macintosh: Devices*.

Summary of AppleTalk Utilities

Pascal Summary

Constants

```
CONST
   setSelfSend          =  256;        {allow intranode delivery, csCode}
   GetATalkInfo         =  258;        {get AppleTalk information, csCode}
```

Data Types

MPP Parameter Block for PSetSelfSend and PGetAppleTalkInfo

```
TYPE MPPParmType =   (...SetSelfSendParm,
                          GetAppleTalkInfoParm...);
TYPE MPPParamBlock =
   PACKED RECORD
      qLink:               QElemPtr;      {reserved}
      qType:               Integer;       {reserved}
      ioTrap:              Integer;       {reserved}
      ioCmdAddr:           Ptr;           {reserved}
      ioCompletion:        ProcPtr;       {completion routine}
      ioResult:            OSErr;         {result code}
      ioNamePtr:           StringPtr;     {reserved}
      ioVRefNum:           Integer;       {reserved}
      ioRefNum:            Integer;       {driver reference number}
      csCode:              Integer;       {primary command code}
   CASE MPPParmType OF
      SetSelfSendParm:
         (newSelfFlag:     Byte;          {self-send toggle flag}
         oldSelfFlag:      Byte);         {previous self-send state}
      GetAppleTalkInfoParm:
         (version:         Integer;       {requested info version}
         varsPtr:          Ptr;           {pointer to MPP variables}
         DCEPtr:           Ptr;           {pointer to MPP DCE}
         portID:           Integer;       {port number [0..7]}
```

```
        configuration:    LongInt;       {32-bit configuration word}
        selfSend:         Integer;       {nonzero if self-send enabled}
        netLo:            Integer;       {low value of network range}
        netHi:            Integer;       {high value of network range}
        ourAddr:          LongInt;       {our 24-bit AppleTalk address}
        routerAddr:       LongInt;       {24-bit address of last router}
        numOfPHs:         Integer;       {maximum number of protocol }
                                         { handlers}
        numOfSkts:        Integer;       {maximum number of static sockets}
        numNBPEs:         Integer;       {maximum concurrent NBP requests}
        ntQueue:          Ptr;           {pointer to registered name queue}
        LAlength:         Integer;       {length in bytes of data-link addr}
        linkAddr:         Ptr;           {data-link address returned}
        zoneName:         Ptr);          {zone name returned}
END;

MPPPBPtr =  ^MPPParamBlock;
```

Routines

Obtaining Information About the .MPP Driver and the Current Network Environment

```
FUNCTION PGetAppleTalkInfo  (thePBptr: MPPPBPtr; async: Boolean): OSErr;
```

Enabling Intranode Delivery of DDP Packets

```
FUNCTION PSetSelfSend       (thePBptr: MPPPBPtr; async: Boolean): OSErr;
```

Getting the Addresses of Your Node and Local Internet Router

```
FUNCTION GetNodeAddress     (VAR myNode: Integer; VAR myNet: Integer): OSErr;
FUNCTION GetBridgeAddress:  Integer;
```

Opening and Closing Drivers

```
FUNCTION MPPOpen:           OSErr;
FUNCTION MPPClose:          OSErr;
FUNCTION IsMPPOpen:         Boolean;
FUNCTION IsATPOpen:         Boolean;
FUNCTION OpenXPP            (VAR xppRefnum: Integer): OSErr;
```

C Summary

Constants

```
/*csCodes/
enum {
   setSelfSend       =     256,     /*intranode packet delivery*/
   GetATalkInfo      =     258      /*get AppleTalk information*/
};
```

Data Types

MPP Parameter Block for PSetSelfSend and PGetAppleTalkInfo

```
union ParamBlockRec {
   MPPparms       MPP;           /*general MPP parms*/
};

typedef MPPParamBlock *MPPPBPtr;

#define MPPATPHeader \
   QElem          *qLink;        /*reserved*/\
   short          qType;         /*reserved*/\
   short          ioTrap;        /*reserved*/\
   Ptr            ioCmdAddr;     /*reserved*/\
   ProcPtr        ioCompletion;  /*completion routine*/\
   OSErr          ioResult;      /*result code*/\
   long           userData;      /*reserved*/\
   short          reqTID;        /*reserved*/\
   short          ioRefNum;      /*driver reference number*/\
   short          csCode;        /*call command code*/

typedef struct {
   MPPATPHeader
   char     newSelfFlag;         /*self-send toggle flag*/
   char     oldSelfFlag;         /*previous self-send state*/
}SetSelfparms;

typedef struct {
   MPPATPHeader
   short    version;          /*requested info version*/
   Ptr      varsPtr;          /*pointer to well-known MPP vars*/
```

```
   Ptr      DCEPtr;           /*pointer to MPP DCE*/
   short    portID;           /*port number [0..7]*/
   long     configuration;    /*32-bit configuration word*/
   short    selfSend;         /*nonzero if self-send enabled*/
   short    netLo;            /*low value of network range*/
   short    netHi;            /*high value of network range*/
   long     ourAdd;           /*our 24-bit AppleTalk address*/
   long     routerAddr;       /*24-bit address of last router*/
   short    numOfPHs;         /*maximum number of protocol handlers*/
   short    numOfSkts;        /*maximum number of static sockets*/
   short    numNBPEs;         /*maximum number of concurrent NBP requests*/
   Ptr      nTQueue;          /*pointer to registered name queue*/
   short    LAlength;         /*length in bytes of data-link addr*/
   Ptr      linkAddr;         /*data-link address returned*/
   Ptr      zoneName;         /*zone name returned*/
}GetAppleTalkInfoParm;

typedef union {
   MPPparms                  MPP;          /*general MPP parms*/
   SetSelfparms              SETSELF;
   GetAppleTalkInfoParm      GAIINFO;
}MPPParamBlock;

typedef MPPParamBlock *MPPPBPtr;
```

Routines

Obtaining Information About the .MPP Driver and the Current Network Environment

```
pascal OSErr PGetAppleTalkInfo
                            (MPPPBPtr thePBptr,Boolean async);
```

Enabling Intranode Delivery of DDP Packets

```
pascal OSErr PSetSelfSend   (MPPPBPtr thePBptr,Boolean async);
```

Getting the Addresses of Your Node and Local Internet Router

```
pascal OSErr GetNodeAddress
                            (short *myNode,short *myNet);
pascal short GetBridgeAddress
                            (void);
```

Opening and Closing Drivers

```
pascal OSErr MPPOpen        (void);
pascal OSErr MPPClose       (void);
pascal Boolean IsMPPOpen    (void);
pascal Boolean IsATPOpen    (void);
pascal OSErr OpenXPP        (short *xppRefnum);
```


Assembly-Language Summary

Constants

Unit Number for the .MPP driver

```
mppUnitNum      EQU      9          ;MPP unit number
mppRefNum       EQU      –10        ;MPP driver reference number
```

Command Codes

```
setSelfSend     EQU      256        ;set to allow writes to self, control call
GetATalkInfo    EQU      258        ;get AppleTalk information, control call
```

Zone and Router Bits

```
BadZoneHintBit EQU       7          ;1, if zone hint was found invalid when the
                                    ; .MPP driver was opened
RouterBit       EQU      30         ;1, if this is a router port
```

MPP Queue Element Standard Structure

```
;arguments passed in the CSParam area
newSelfFlag     EQU      $1C        ;offset, new value for self-send flag
oldSelfFlag     EQU      $1D        ;old value of self-send flag
```

GetAppleTalkInfo

```
GAIVersion      EQU      1         ;highest version for GAI params
```

Data Structures

MPP Parameter Block Common Fields for PGetAppleTalkInfo and PSetSelfSend

0	qLink	long	reserved
4	qType	word	reserved
6	ioTrap	word	reserved
8	ioCmdAddr	long	reserved
12	ioCompletion	long	address of completion routine
16	ioResult	word	result code
18	ioNamePtr	long	reserved
22	ioVRefNum	word	reserved
24	ioRefNum	word	driver reference number

GetAppleTalkInfo Parameter Variant

16	ioResult	word	result code
26	csCode	word	command code; always `GetAppleTalkInfo`
28	version	word	version of function
30	varsPtr	long	pointer to the .MPP driver variables
34	DCEPtr	long	pointer to DCE for the .MPP driver
38	portID	word	port number
40	configuration	long	configuration flags
44	selfSend	word	nonzero if self-send is enabled
46	netLo	word	low value of network range
48	netHi	word	high value of network range
50	ourAddr	long	local 24-bit AppleTalk address
54	routerAddr	long	24-bit address of router
58	numOfPHs	word	maximum number of protocol handlers
60	numOfSkts	word	maximum number of static sockets
62	numNBPEs	word	maximum number of concurrent NBP requests
64	ntQueue	long	pointer to registered names table
68	LAlength	word	length in bytes of data-link address (extended networks only)
70	linkAddr	long	pointer to data-link address buffer (extended networks only)
74	zoneName	long	pointer to zone name buffer

PSetSelfSend Parameter Variant

26	csCode	word	always `setSelfSend`
28	newSelfFlag	byte	flag that turns intranode delivery on or off
29	oldSelfFlag	byte	flag that reports the previous state of intranode delivery, whether it was on or off

Result Codes

noErr	0	No error
paramErr	–50	Version number is too high
portInUse	–97	Driver open error code indicating that the port is in use
portNotCf	–98	Driver open error code indicating that the parameter RAM is not configured for this connection
noMPPErr	–3102	The .MPP driver is not installed