The Link-Access Protocol (LAP) Manager is a set of operating-system utilities that provide a standard interface between the higher-level AppleTalk protocols and the various link-access protocols, such as LocalTalk (LLAP), EtherTalk (ELAP), TokenTalk (TLAP), and FDDITalk (FLAP). This chapter describes the LAP Manager programming interfaces to the AppleTalk Transition Queue and the 802.2 packet protocol handlers only. This chapter does not discuss the LAP Manager interface to AppleTalk connection files of type `'adev'` that comprise the data links. Apple Computer, Inc. recommends that you not write your own `'adev'` files. However, for a description of the LAP Manager that includes the interface to AppleTalk connection files for EtherTalk and other AppleTalk connections, see the *Macintosh AppleTalk Connections Programmer's Guide*.

You should read this chapter if you want the LAP Manager to notify you when a 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. This chapter also describes how you can define a transition to notify other applications of a transition event that your application effects.

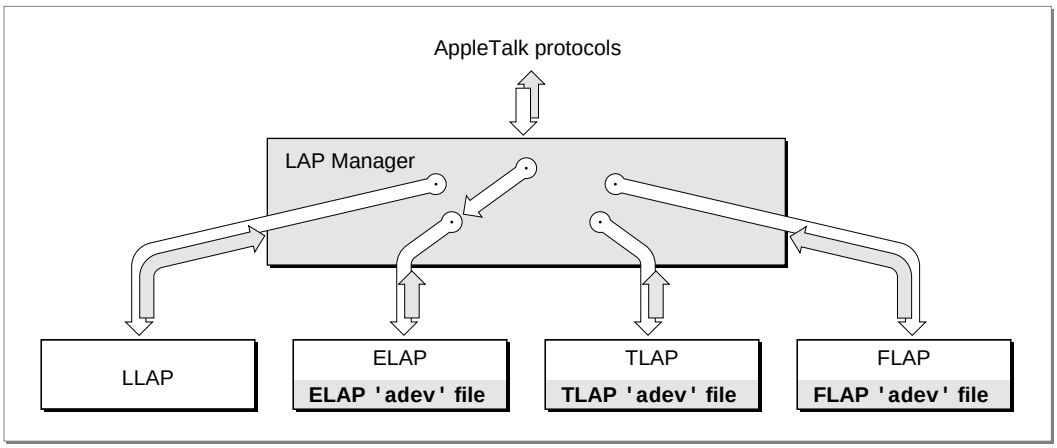
You should also read this chapter if your application processes 802.2 Type 1 packets. In this case, you must write a protocol handler that reads 802.2 Type 1 data packets and install your protocol handler as a client of the LAP Manager.

For an overview of the LAP Manager and how it fits within the AppleTalk protocol stack, read the chapter "Introduction to AppleTalk" in this book, which also introduces and defines some of the terminology used in this chapter. For additional information on the IEEE 802.2 standard, see *Inside AppleTalk*, second edition.

About the LAP Manager

A Macintosh computer on an AppleTalk network can include one or more AppleTalk connection files. An ***AppleTalk connection file*** is a file of type `'adev'` that contains a link-access protocol implementation for a data link (ELAP for EtherTalk, for example). One important function of an AppleTalk connection file is to implement the AppleTalk Address Resolution Protocol (AARP) that maps hardware layer addresses to AppleTalk node addresses. The LAP Manager makes it possible for the user to select among AppleTalk connection files by using the Network control panel to specify which network is to be used for the node's AppleTalk connection. When the user selects a connection from the Network control panel, the LAP Manager routes AppleTalk communications through the selected link-access protocol and hence through the selected hardware. The LAP Manager acts as a switching mechanism, interceding between the higher-level AppleTalk protocols and the data links so that when a user selects or changes the type of data link to be used, the process is transparent to the higher-level AppleTalk protocols and has no effect on applications that are clients of these protocols. Figure 10-1 shows this service that the LAP Manager provides. This figure does not show an AppleTalk connection file for LLAP because AARP is not used for LLAP and address mapping is not necessary.

Figure 10-1 LAP Manager connecting the higher-level AppleTalk protocols with the selected data link



In addition to providing an interface to AppleTalk connection files, the LAP Manager also maintains the ***AppleTalk Transition Queue,*** which is an operating-system queue that can notify your application each time an AppleTalk transition occurs. An ***AppleTalk transition*** is an event, such as an AppleTalk driver being opened or closed or a network connection being broken, that can affect your AppleTalk application.

At any given time there might be two or more applications running that use AppleTalk. If one of these applications opens the .MPP driver, the other AppleTalk applications that use the driver are affected. If the operating system closes the AppleTalk .MPP driver, all AppleTalk applications using the driver are affected. To ensure that your application is not adversely affected by such an event, your application can place an entry in the AppleTalk Transition Queue. The LAP Manager sends a message to each entry each time the operating system or any routine performs any of these operations:

- opens the .MPP driver
- closes the .MPP driver
- indicates that it intends to close the .MPP driver
- cancels its intention to close the .MPP driver
- reports that it is changing the flagship name (This is a personalized name that a user can enter to identify the system when it is connected to an AppleTalk network.)
- indicates that it intends to change the flagship name
- cancels its intention to change the flagship name
- reports that the network connectivity has changed (for example, that a previously interconnected network is no longer available)
- reports that the cable range for the current network has been changed
- changes the speed of the CPU
- defines its own AppleTalk event and calls the AppleTalk Transition Queue to inform it that such an event occurred

Each of these events is referred to as an *AppleTalk transition*.

The LAP Manager also includes a protocol handler that reads 802.2 packets and provides an interface that allows you to attach your own ***protocol handler*** to receive 802.2 Type 1 packets. An 802.2 protocol handler is an application or process that receives, reads, and processes these 802.2 data packets. An 802.2 packet conforms to the 802.2 data-link standard called ***Logical Link Control (LLC)*** defined by the Institute of Electrical and Electronics Engineers (IEEE) for use on Ethernet, token ring, FDDI, and certain other data links. The 802.2 Type 1 protocol specifies a *connectionless* or *datagram* service. (The AppleTalk ELAP, TLAP, and FLAP implementations process 802.2 Type 1 packets.)

Using the LAP Manager

This section describes how you can use the LAP Manager's AppleTalk Transition Queue. Then it describes how to attach and detach protocol handlers for 802.2 Type 1 data packets using the `L802Attach` and `L802Detach` routines.

To use the AppleTalk Transition Queue, you add an entry for your application that contains a pointer to a transition event handler routine that you must provide to receive notification of transitions and to perform any additional processing that you want to perform in reaction to the transition.

After you add your entry, the LAP Manager will call your transition event handler routine to notify you that an AppleTalk transition either is about to occur or has occurred. The description of how to use the AppleTalk Transition Queue includes

- how to determine if the LAP Manager is installed on the node running your application
- how to add an entry to the AppleTalk Transition Queue
- how to write the routine that you must provide that the LAP Manager calls to notify you of the transition
- how to handle each of the standard AppleTalk transitions that can occur and about which your routine will be notified
- how to handle developer-defined transitions
- how to define your own transition events

Determining if the LAP Manager Is Installed

Before you issue any calls to the LAP Manager, you should check to determine if the LAP Manager is installed on the node that is running your application. The LAP Manager is implemented beginning with AppleTalk version 53. To determine if the LAP Manager is installed, you can check the low-memory global variable `LAPMgrPtr`. However, Apple Computer, Inc. recommends that you use a higher-level method to perform this check, such as the one that the code in Listing 10-1 shows.

Listing 10-1 Checking to determine if the LAP Manager is installed

```
FUNCTION GestaltAvailable: Boolean;
CONST
   _Gestalt = $A1AD;
BEGIN
   GestaltAvailable := TrapAvailable(_Gestalt);
END;

FUNCTION AppleTalkVersion: Integer;
CONST
   versionRequested = 1;   {version of SysEnvRec}
VAR
   refNum: Integer;
   world: SysEnvRec;
   attrib: LongInt;
BEGIN
   AppleTalkVersion := 0;   {default to no AppleTalk}
   IF OpenDriver('.MPP', refNum) = noErr THEN
                           {open the AppleTalk driver}
      IF GestaltAvailable THEN
      BEGIN
         IF (Gestalt(gestaltAppleTalkVersion, attrib) = noErr)
            THEN
            AppleTalkVersion := BAND(attrib, $000000FF);
      END
      ELSE {Gestalt or gestaltAppleTalkVersion selector isn't }
            { available.}
         IF SysEnvirons(versionRequested, world) = noErr THEN
            AppleTalkVersion := world.atDrvrVersNum;
END;

FUNCTION LAPMgrExists: Boolean;
BEGIN
   {AppleTalk Phase 2 is AppleTalk version 53 and later}
   LAPMgrExists := (AppleTalkVersion >= 53);
END;
```

Here is the declaration for the `TrapAvailable` function that the code in Listing 10-1 calls:

```
FUNCTION TrapAvailable (theTrap: Integer): Boolean;
VAR
   tType: TrapType;
BEGIN
   tType := GetTrapType(theTrap);
   IF tType = ToolTrap THEN
   BEGIN
      theTrap := BAND(theTrap, $07FF);
      IF theTrap >= NumToolboxTraps THEN
         theTrap := _Unimplemented, ToolTrap;
END;
```

Adding an Entry to the AppleTalk Transition Queue

To ensure that your application is not adversely affected by a transition event, your application places an entry in the AppleTalk Transition Queue.

To do this, you must create an AppleTalk Transition Queue entry record of type `ATQentry` and give the LAP Manager a pointer to it. See "The AppleTalk Transition Queue Entry" on page 10-33 for a description of the AppleTalk Transition Queue entry record. This record includes a `CallAddr` field that holds a pointer to a ***transition event handler routine*** that you provide, which is described in the following section "How the LAP Manager Calls Your Transition Event Handler Routine."

Because you provide the memory for the queue entry, you can add as many fields to the end of the entry as you wish for your own purposes. Whenever the LAP Manager calls your transition event handler routine, it provides you with a pointer to the queue entry so that you can have access to the information you stored at the end of your queue entry.

After you have created the AppleTalk Transition Queue entry record, you use the `LAPAddATQ` function to add the entry to the AppleTalk Transition Queue. You pass a pointer to the entry record as the value of the function's `theATQEntry` parameter. Listing 10-2 shows how to do this using assembly language: you place a routine selector in the D0 register, place a pointer to your AppleTalk Transition Queue entry in the A0 register, and execute a JSR instruction to an offset past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18). The offset to the LAP Manager routines is given by the constant `LAPMgrCall` (2).

Listing 10-2 Adding an AppleTalk Transition Queue entry

```
LAPMgrPtr   EQU     $B18            ;entry point for LAP Manager
LAPMgrCall  EQU     2               ;offset to LAP Manager
                                    ; routines
ATQEntry    EQU     *               ;pointer to ATQ entry

            MOVEQ   #23,D0          ;place routine selector
                                    ; in D0
            MOVE.L  LAPMgrPtr,An    ;put pointer to LAP Mgr in An
            MOVE.L  ATQEntry,A0     ;put ATQ entry in A0
            JSR     LAPMgrCall(An)  ;jump to start of LAP Mgr
                                    ; routines
```

When you no longer want to be notified of transition events or before your program exits, you use the `LAPRmvATQ` function to remove your AppleTalk Transition Queue entry from the queue. Listing 10-3 shows how to do this from assembly language; you place the routine selector in the D0 register, place a pointer to your AppleTalk Transition Queue entry in the A0 register, and execute a JSR instruction to an offset past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18). The offset to the LAP Manager routines is given by the constant `LAPMgrCall` (2).

Listing 10-3 Removing an AppleTalk Transition Queue entry

```
LAPMgrPtr   EQU     $B18            ;entry point for LAP Manager
LAPMgrCall  EQU     2               ;offset to LAP Manager
                                    ; routines
ATQEntry    EQU     *               ;pointer to ATQ entry

            MOVEQ   #24,D0          ;place routine selector
                                    ; in D0 (24 to remove an
                                    ; entry)
            MOVE.L  LAPMgrPtr,An    ;put pointer to LAP Mgr in An
            MOVE.L  ATQEntry,A0     ;put ATQ entry in A0
            JSR     LAPMgrCall(An)  ;jump to start of LAP Mgr
                                    ; routines
```

How the LAP Manager Calls Your Transition Event Handler Routine

This section describes how to write a transition event handler routine that responds to notification of AppleTalk transitions. Because the LAP Manager calls your transition event routine using C conventions, a transition event handler routine written in Pascal requires glue code to function correctly. To help solve this problem, this section includes a discussion of how to write a transition event routine using Pascal, and it also includes glue code that you will need. This section also describes the standard AppleTalk transitions and how your routine can respond to a particular transition.

When you have used the `LAPAddATQ` function to add an entry to the AppleTalk Transition Queue, the LAP Manager calls the transition event handler routine, whose pointer you pass to the LAP Manager in the AppleTalk Transition Queue entry record, whenever an AppleTalk transition occurs.

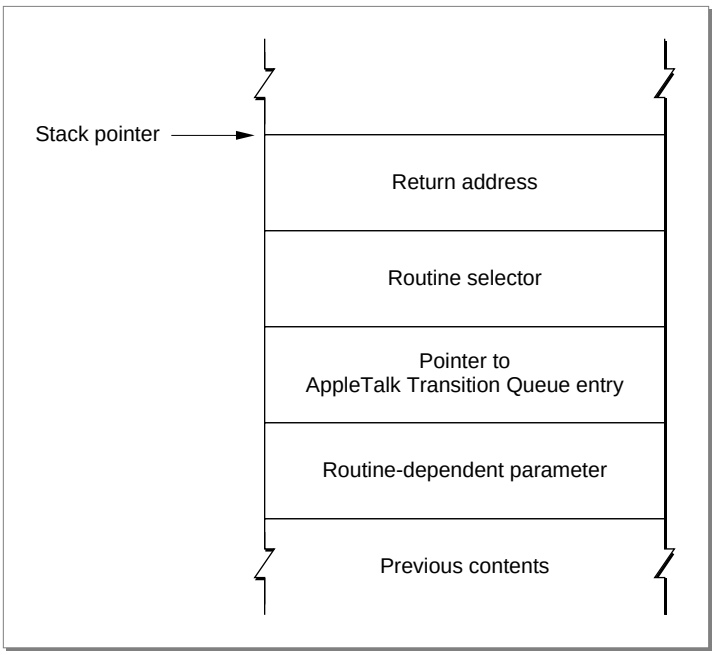
Table 10-1 shows the standard AppleTalk transitions (each of which is discussed later in this section) and their constants and routine selectors.

Table 10-1 AppleTalk transitions and their constants and routine selectors

AppleTalk transition	Constant	Routine selector
Open	`ATTransOpen`	0
Prepare-to-close	`ATTransClose`	2
Permission-to-close	`ATTransClosePrep`	3
Cancel-close	`ATTransCancelCATTransCancelClose`	4
Network-connection-change	`ATTransNetworkTransition`*	5
Flagship-name-change	`ATTransNameChangeTellTask`*	6
Permission-to-change-flagship-name	`ATTransNameChangeAskTask`*	7
Cancel-flagship-name-change	`ATTransCancelNameChange`*	8
Cable-range-change	`ATTransCableChange`*	`'rnge'`
CPU-speed-change	`ATTransSpeedChange`*	`'sped'`

* The constants marked with an asterisk are not included in the header files; you can use the routine selectors for these transitions, or you can define the constants in your application.

From assembly language, when the LAP Manager calls your routine, the stack looks like this:



The first item on the stack (after the 4-byte-long return address) is a routine selector. There is one routine selector for each type of transition. Some transition events have a single-digit routine selector. Other transition events are four-character codes. Codes starting with an uppercase letter (A through Z) are reserved for use by developers. All other codes are reserved for use by Apple Computer, Inc.

The second item passed to your routine on the stack is a pointer to your routine's entry in the AppleTalk Transition Queue. You can use this pointer to gain access to any fields at the end of the queue entry that you allocated for your own use. The last item passed to your routine on the stack is a 4-byte-long parameter whose meaning depends on the type of transition.

With the exception of the open transition, the prepare-to-close transition, the flagship-name-change transition, the permission-to-change-flagship-name transition, and the cancel-flagship-name transition, the interface between the AppleTalk Transition Queue and your routine must follow these conventions:

- Your routine must preserve all registers except D0, D1, D2, A0, and A1.
- All parameters are passed on the stack as long words.
- Because your routine might be called at interrupt time, your routine must not make any direct or indirect calls to the Memory Manager, and it cannot depend on handles to unlocked blocks being valid, unless otherwise noted in the description of the transition event.

- If you want to use any of your application's global variables, you must save the contents of the A5 register before using the variables and you must restore the A5 register before your routine terminates.

Again, these restrictions do not apply to the open transition, the prepare-to-close transition, and the three flagship-name transitions.

IMPORTANT

It is important that you return a 0 in the D0 register whenever you receive a transition event routine selector that you do not recognize or do not choose to handle. Returning a nonzero value in the D0 register might cause the system to cancel an attempt to close AppleTalk, for example, or it might be misinterpreted in some other way. You should only return a nonzero result to known transition events. ▲

Writing a Transition Event Handler Routine Using Pascal

The LAP Manager assumes that you will use the `CallAddr` field of your event record to pass it a pointer to a transition event handler routine that is written in the C programming language. The LAP Manager use C calling conventions when it calls your routine.

If you write your transition event handler routine in Pascal, you must include a glue code wrapper routine. You can use either the sample glue code provided in this section or your own method. To use this glue code, you must modify the AppleTalk Transition Queue entry record to include a field to hold a pointer to your Pascal transition event handler routine. You must add this field directly after the `CallAddr` field. You use the `CallAddr` field to pass the address of the assembly-language glue code routine. Here is the type declaration for an AppleTalk Transition Queue entry record that includes the additional field that is required if you use the glue code:

```
TYPE myATQEntry =
RECORD
   qlink:         Ptr;              {ptr to next queue entry}
   qType:         Integer;          {reserved}
   CallAddr:      ProcPtr;          {ptr to the glue code}
   PATQProcPtr:   ProcPtr;          {ptr to Pascal ATQ }
                                    { routine; this field must }
                                    { follow the CallAddr field. }
                                    { Do not change the order of }
                                    { these fields.}
   globs:         TransEventPtr;    {ptr to user defined globals}
END;

myATQEntryPtr = ^myATQEntry;
myATQEntryHdl = ^myATQEntryPtr;
```

The following segment of code shows how to add an AppleTalk Transition Queue entry to the queue. In this example, the actual transition event handler routine is called `ATQueueProc`. The glue code routine is called `CallTransQueue`. The `LAPAddATQ` function passes the glue code routine to the LAP Manager in the `CallAddr` field of the AppleTalk Transition Queue entry `myATQEntry`.

```
VAR
   gATQEntry: myATQEntry;
   OSErr: err;

BEGIN
   gATQEntry.CallAddr := ProcPtr(@CallTransQueue);
   gATQEntry.PATQProcPtr := ProcPtr(@ATQueueProc);
   err := LAPAddATQ(ATQEntryPtr(@gATQEntry));
```

Listing 10-4 shows the sample assembly-language glue code routine `CallTransQueue` that you can use if you write your transition event handler routine in Pascal. The glue routine takes the parameters from the stack and sets up a Pascal stack, then calls the function pointed to by the `PATQProcPtr` field of the AppleTalk Transition Queue entry record. On return, the glue code pulls the result from the stack and puts it into the D0 register, where the LAP Manager expects to find it.

Listing 10-4 Glue code for a Pascal transition event handler routine

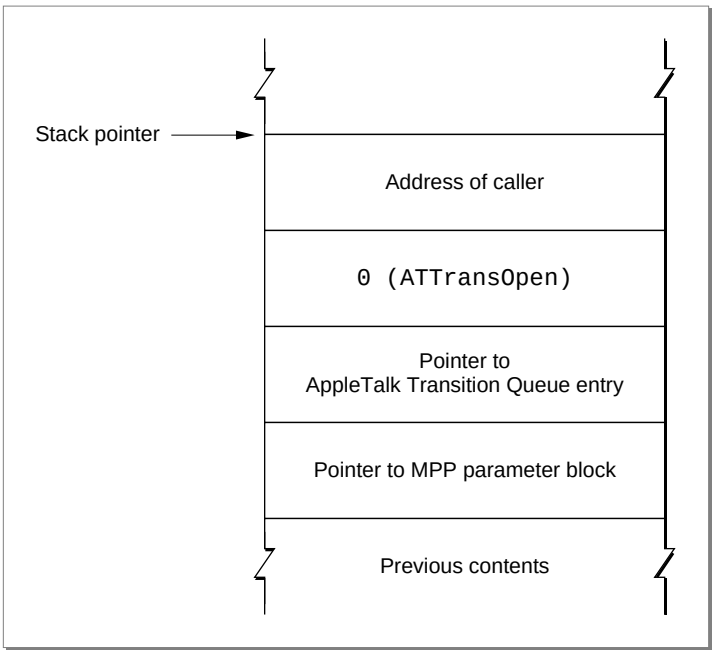
```
;FUNCTION CallTransQueue (selector: LongInt; q: ATQEntryPtr;
;                         p: Ptr): LongInt;
;EXTERNAL;

CallTransQueue PROCEXPORT
   LINK       A6,#$0000       ;set up a local stack frame
   CLR.L      -(A7)           ;set space for return result
   MOVE.L     $0008(A6),-(A7) ;move selector to stack
   MOVE.L     $000C(A6),-(A7) ;move ATQPtr to stack
   MOVEA.L    (A7),A0         ;put copy ATQPtr in A0
   MOVEA.L    $000A(A0),A0    ;put pointer to real ATQ in A0
   MOVE.L     $0010(A6),-(A7) ;move last parameter:
                              ; pointer to stack
   JSR        (A0)            ;call the Pascal ATQ function
   MOVE.L     (A7)+,D0        ;move result into D0
   UNLK       A6              ;tear down local stack frame
   RTS                        ;return
   ENDP
   END
```

Open Transition

When an application calls the `MPPOpen` function or the Device Manager's `OpenDriver` function, AppleTalk attempts to open the .MPP driver. If the .MPP driver is already open, the LAP Manager does not call the AppleTalk Transition Queue transition event handler routines. If AppleTalk successfully opens the .MPP driver, the LAP Manager then calls every routine listed in the AppleTalk Transition Queue with an open transition (`ATTransOpen`).

When the LAP Manager calls your transition event handler routine, the stack looks like this:



The last item on the stack for an open transition is a pointer to the start of the Device Manager extended parameter block used by the routine that opened the .MPP driver. This pointer is provided for your information only; you must not change any of the fields in this parameter block.

Your transition event handler routine can perform any tasks you wish in response to the notification that the .MPP driver has been opened, such as using the Name-Binding Protocol (NBP) to register a name on the internet. Return 0 in the D0 register to indicate that your routine executed with no error.

Note
The open transition event occurs at system task time, during which you can allocate memory. ◆

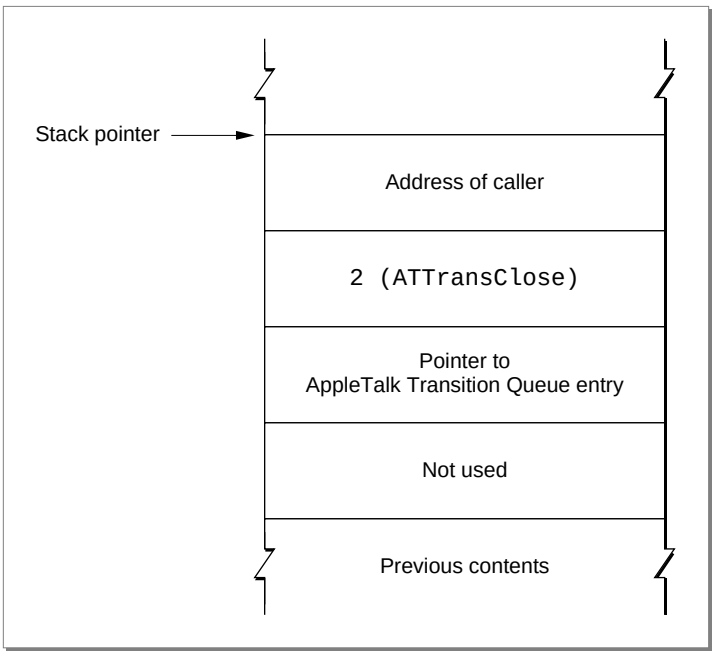
Prepare-to-Close Transition

When any routine calls the `MPPClose` function or the Device Manager's `CloseDriver` function to close the .MPP driver, the LAP Manager calls every routine listed in the AppleTalk Transition Queue before the .MPP driver closes with an `ATTransClose` transition; if the .MPP driver is already closed when a routine calls either `MPPClose` or `CloseDriver`, the LAP Manager does not call the transition event handler routines in the AppleTalk Transition Queue.

When the system closes the .MPP driver

Whereas it is unlikely that opening the .MPP driver will adversely affect another program, an application should never close the .MPP driver because another program might be using it. Under certain circumstances, however, the system might close the .MPP driver, for example, when the user changes the network connection. In this case, the system will send a permission-to-close transition to each routine in the AppleTalk Transition Queue. This transition indicates that the system intends to close the .MPP driver, and in this way, each transition event handler routine in the queue has the opportunity to deny it permission to do so. When the system sends the permission-to-close transition, any routine in the AppleTalk Transition Queue that wishes to deny permission to close the .MPP driver can return a pointer to a Pascal string that gives the name of the application that placed the entry in the queue. If any routine denies permission to close the .MPP driver, the LAP Manager sends a cancel-close transition to every routine in the AppleTalk Transition Queue that previously received the permission-to-close transition. The application that caused the system to send a permission-to-close transition application may display a dialog box informing the user that another application is using the .MPP driver and showing the name (if any) returned by the transition event handler routine. The dialog box gives the user the option of canceling the request to close AppleTalk or of closing AppleTalk anyway. If the user chooses to close AppleTalk despite the fact that an application is using it, the system calls the `MPPClose` function. The LAP Manager then sends a prepare-to-close transition to each application in the AppleTalk Transition Queue, informing each one that AppleTalk is about to close. In this case, your transition event handler routine must prepare for the imminent closing of AppleTalk; it cannot deny permission to the `MPPClose` function. ◆

When the LAP Manager calls your transition event handler routine, the stack looks like this:



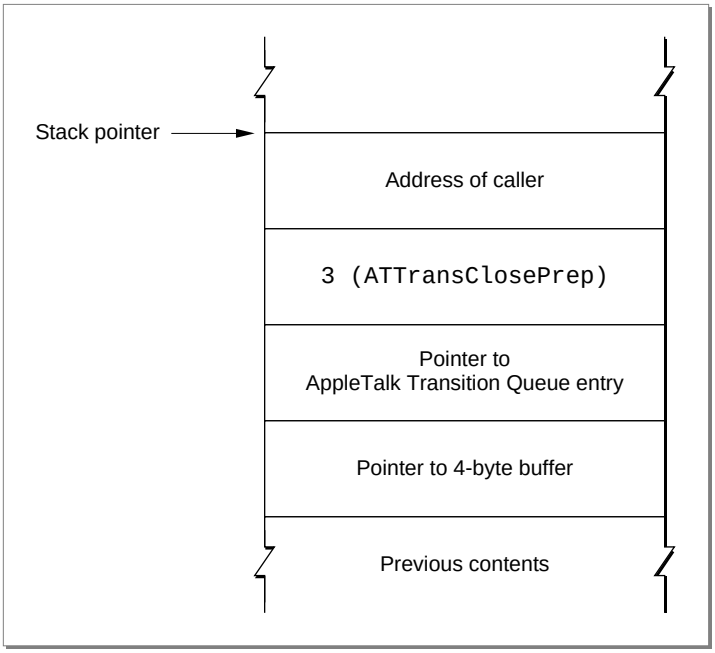
Your routine can perform any tasks you wish to prepare for the imminent closing of AppleTalk, such as ending a session with a remote terminal and informing the user that the connection is being closed. You must return control to the LAP Manager as quickly as possible. Return 0 in the D0 register to indicate that your routine executed with no error.

Note
When the LAP Manager calls your routine with a prepare-to-close transition (that is, a routine selector of `ATTransClose`), you cannot prevent the .MPP driver from closing. ◆

Permission-to-Close Transition

When a routine calls AppleTalk to inform AppleTalk that it wants to close the .MPP driver, the LAP Manager calls every transition event handler routine to request permission to close the .MPP driver with an `ATTransClosePrep` transition.

When the LAP Manager calls your transition event handler routine, the stack looks like this:



The last parameter on the stack is a pointer to a 4-byte buffer. If you intend to deny the request to close the .MPP driver, you place in the buffer a pointer to a Pascal string containing the name of your application. This string belongs to the LAP Manager until the LAP Manager finishes processing the cancel-close transition. The routine that issued the request to close the .MPP driver can then display a dialog box telling the user the name of the application that is currently using AppleTalk.

Your routine can return either a function result of 0 in the D0 register, indicating that it accepts the request to close, or a 1 in the D0 register, indicating that it denies the request to close. Note that the operating system might elect to close the .MPP driver anyway; for example, if the user grants permission to close in response to a dialog box.

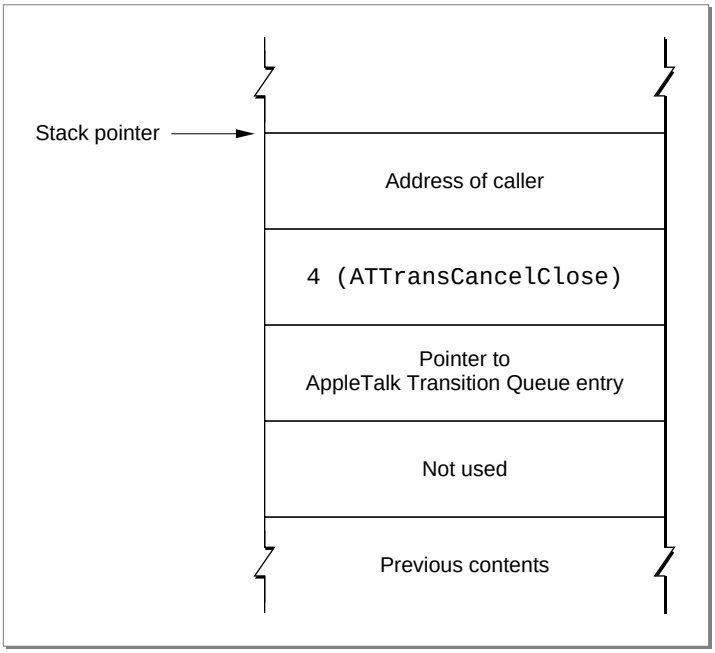
Because the LAP Manager calls your routine again (with the routine selector set to `ATTransClose`) before the .MPP driver actually closes, it is not necessary for your routine to do anything other than grant or deny permission in response to being called for a permission-to-close transition. However, you might want to prohibit users from opening new sessions or establishing new connections while you are waiting for the .MPP driver to close.

Note
Earlier versions of *Inside Macintosh* referred to the `PATalkClosePrep` function as a means of requesting permission to close the .MPP driver. The `PATalkClosePrep` function is now only used internally by the .MPP driver. ◆

Cancel-Close Transition

When any routine in the AppleTalk Transition Queue denies permission for the .MPP driver to close, the LAP Manager calls each routine that has already received the permission-to-close transition with an `ATTransCancelClose` transition to inform it that the request to close the .MPP driver has been canceled.

When the LAP Manager calls your transition event handler routine, the stack looks like this:



If your routine performed any tasks to prepare for the closing of AppleTalk, it should reverse their effects when it is called with the routine selector set to `ATTransCancelClose`. Return 0 in the D0 register to indicate that your routine executed with no errors.

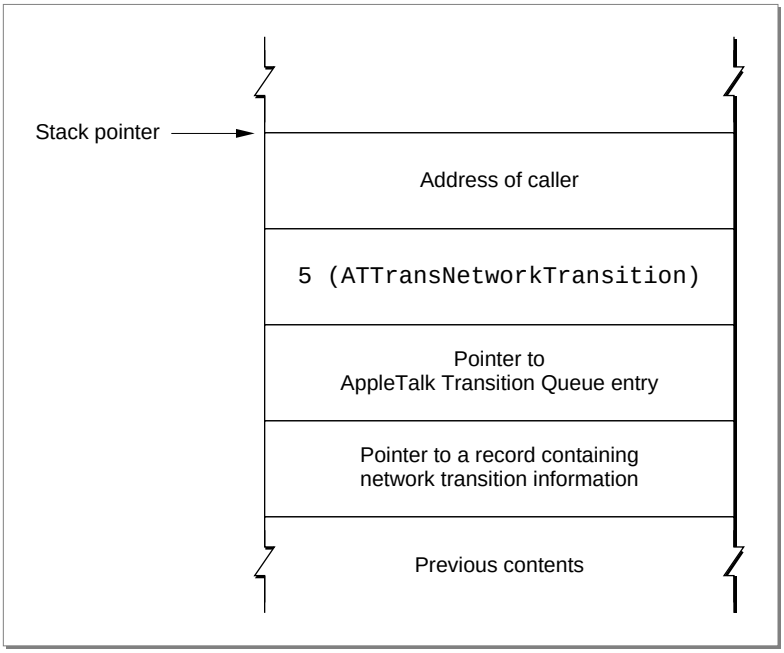
Network-Connection-Change Transition

To receive notification of network connection changes or transitions, your application should process `ATTransNetworkTransition` transitions. All applications running on an AppleTalk network should handle this event, but especially those applications that use multinode IDs.

For example, Apple Remote Access (ARA), which uses multinode architecture, allows a user to establish a connection between two Macintosh computers over standard telephone lines. If the Macintosh that the user dials into is on an AppleTalk network, such as LocalTalk or EtherTalk, the Macintosh effectively becomes a node on that network, and all of the services on that network become available to the user. Because of this relationship, any application that establishes an ARA connection needs to be notified when new AppleTalk connections are established or broken.

Note
Both the AppleTalk Session Protocol (ASP) and the AppleTalk Data Stream Protocol (ADSP) have been modified to respond to network-connection-change transitions. When the AppleTalk drivers that implement these protocols receive notification of a network disconnect transition, they close down sessions on the remote side of the connection. ◆

When the LAP Manager calls your transition event handler routine, the stack looks like this:



Note
If you want to use the constant `ATTransNetworkTransition` for this transition event, you must first declare it in your application because it is not defined in the MPW interface files. ◆

When the LAP Manager calls your routine, the last parameter on the stack contains a pointer to a record that contains a pointer to a network validation procedure. The process that sends notification of the network connection change uses this record to pass to the transition event handler routines a pointer to the network validation procedure; the transition event handler routines can then use this procedure to determine which networks are no longer connected, which networks remain connected, and which new

networks have been added. To read the data in the record that this field points to, you must declare the following record type in your application:

```
TNetworkTransition =
   RECORD
      private:          Ptr;     {pointer used internally by ARA}
      netValidProc:     ProcPtr; {pointer to the network }
                                 { validation procedure}
      newConnectivity:  Boolean; {TRUE = new connectivity, }
                                 { FALSE = loss of connectivity}
   END;
```

You cannot access a `ProcPtr` directly from Pascal. Therefore, if you write your application in Pascal and you want to handle the `ATTransNetworkTransition` event, you need to include the following glue code so that you can access the network validation procedure pointed to by the `netValidProc` field. Listing 10-5 shows the `CallNetValidProc` function glue code that you can use to call the `netValidProc` validation procedure passed in the `TNetworkTransition` record.

Listing 10-5 Glue code to handle the network-connection-change transition from Pascal

```
FUNCTION CallNetValidProc (netTrans: TNetworkTransitionPtr;
                           theNet: LongInt; p: ProcPtr): LongInt;

INLINE
$205F, { MOVEA.L  (SP)+,A0 ;get ProcPtr into A0, and make stack
                           ; right for call }
$4E90; { JSR      (A0)      ;call ProcPtr, and return to caller}
```

The code in Listing 10-6 demonstrates the calling sequence of events for the `CallNetValidProc` glue code.

Listing 10-6 Using the glue code for the network validation procedure

```
CASE selector OF
ATTransNetworkTransition:
BEGIN
   myTNetworkTransitionPtr := TNetworkTransitionPtr(p);
   if (myTNetworkTransitionPtr^.newConnectivity) THEN
   BEGIN
   {
      /*Determine if there is a new connection.*/
   }
   END
```

```
   ELSE
BEGIN
{
   /*If there is a new connection, determine which network */
   /* address needs to be validated and assign the value to */
   /* checkThisNet.*/
}
   checkThisNet = $1234FD00;
                  /*network $1234, node $FD, socket not used*/
   if (CallNetValidProc(myTNetworkTransitionPtr, checkThisNet
   myTNetworkTransitionPtr^.netValidProc) <> 0) THEN

   /*Take the appropriate action depending on result.*/
```

Apple Remote Access (ARA) is an example of a process that generates network-connection-change transitions to inform transition event handler routines and resident processes that network connectivity has changed. ARA uses the `TNetworkTransition` record to inform the routines about the changes. The `newConnectivity` field of the `TNetworkTransition` record identifies the type of change that has occurred:

- If this flag is `TRUE`, the network that your node is connected to through ARA has connected to a new internet. In this case, the LAP Manager will return all network addresses identifying them as reachable.
- If this flag is `FALSE`, specific networks are no longer reachable.

Because ARA is connection oriented, it can identify the location of a specific network and inform transition event handler routines that a network is no longer reachable. You can use this information to identify the loss of connections immediately instead of waiting to discover that the other end of the connection is no longer responding.

The `netValidProc` field of the `TNetWorkTransition` record contains a network validation hook for a function that you can use to query ARA about a specific network to determine if that network is still reachable. If the network is reachable, the validation function returns `TRUE`. You can call this function repeatedly to determine the status of each network that you are interested in. If you use the Pascal language to write your transition event handler routine, you must implement glue code to use the network validation procedure.

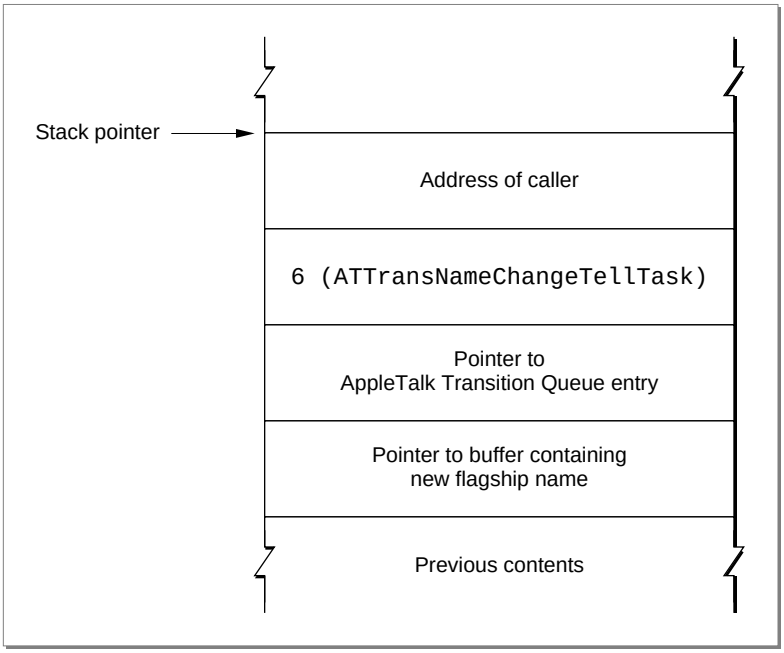
The information that the validation function returns is valid only for those routines that use the function in response to a network-connection-change transition.

Note
A network-connection-change transition can be sent at interrupt time. Because of this, you should follow the conventions that apply when a routine is called during an interrupt. For example, your routine should not call routines that move memory and you should not call AppleTalk functions synchronously. ◆

Flagship-Name-Change Transition

System 7 allows a user to enter a personalized name that identifies the system when it is connected to an AppleTalk network. This is called the ***flagship name.*** An application that provides network services for a workstation should use the flagship name so that the user can personalize the name that identifies the workstation to the network while reserving the use of the Chooser name for server connection identification. If your application utilizes flagship names, your routine should process `ATTransNameChangeTellTask` transitions. When the LAP Manager calls your routine with an `ATTransNameChangeTellTask` transition, you cannot prevent the flagship name from being changed.

When a routine calls the `ATEvent` procedure to change the flagship name, the LAP Manager calls every routine listed in the AppleTalk Transition Queue with an `ATTransNameChangeTellTask` transition. When the LAP Manager calls your transition event handler routine, the stack looks like this:

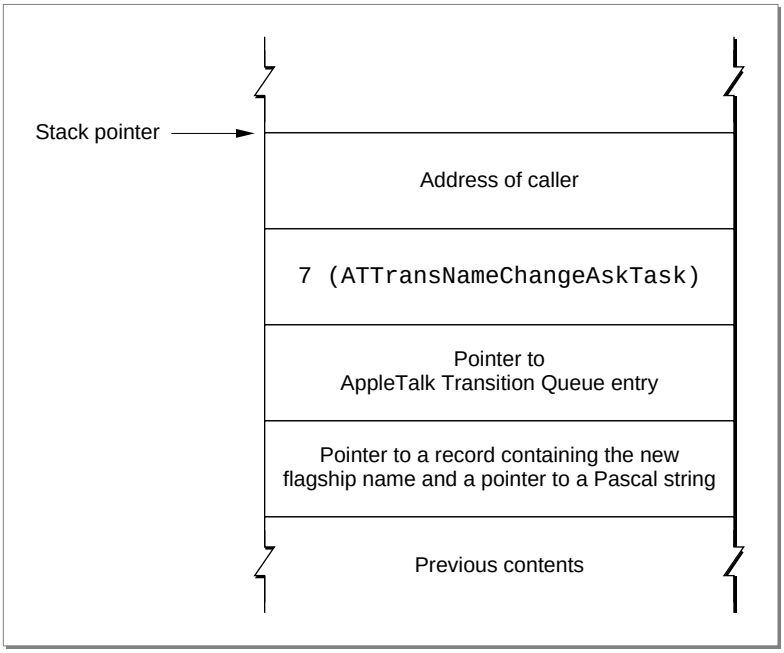


The last item on the stack is a pointer to a Pascal string that is the new flagship name to be registered. Your routine should remove the NBP registrations of entities under the old flagship name. You can make synchronous calls to NBP to remove a registered entity. Return a result of 0 in the D0 register to indicate that your routine executed with no error.

Note
Your application should only respond to flagship name changes about which it receives notification. Do not attempt to change the flagship name. ◆

Permission-to-Change-Flagship-Name Transition

If your application utilizes flagship names, your transition event handler routine should process `ATTransChangeNameAskTask` transitions. When a process makes a request to change the flagship name, the LAP Manager calls every routine listed in the AppleTalk Transition Queue with an `ATTransChangeNameAsk` transition to request permission to change the name. When the LAP Manager calls your transition event handler routine, the stack looks like this:



The last item on the stack contains a pointer to a record that holds the new flagship name. The `NameChangeInfo` record also includes a field that you use to identify your application if you deny the name-change request. To read from and write to the record, you must declare the following record type in your application:

```
NameChangeInfo =
   RECORD
      newObjStr:  Str32;      {new flagship name}
      name:       StringPtr;  {pointer to }
   END;                       { application's name}
```

The `newObjStr` field contains the proposed flagship name change. Your routine can inspect the `newObjStr` field. If your routine denies the name-change request, you must provide as the value of the `name` field a pointer to a buffer containing a Pascal

string that names your application. The LAP Manager returns this pointer to the process that requested the flagship name change so that the process can then display a dialog box telling the user the name of the application that refused the name change.

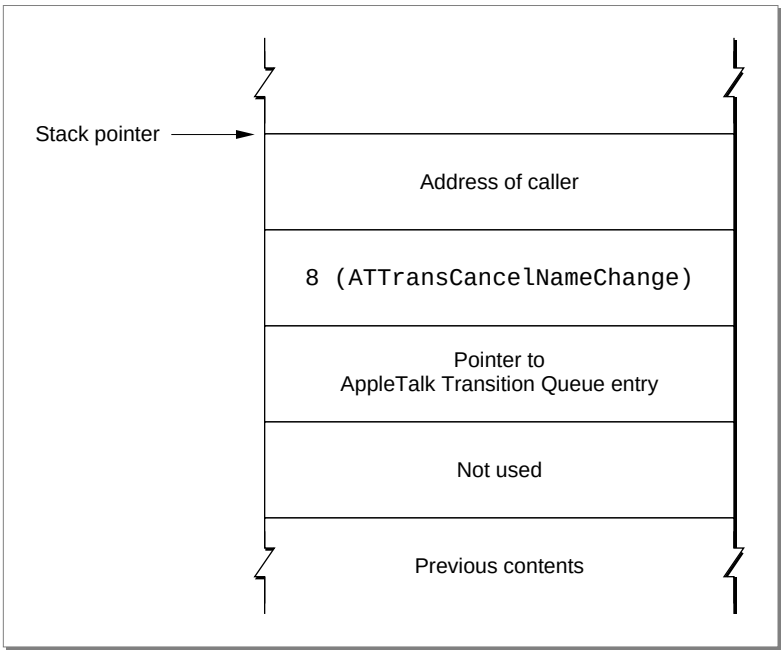
If your application does not deny the request, you can make synchronous calls to NBP to attempt to register your application under the new flagship name while your transaction event handler routine is processing the request. Apple Computer, Inc. recommends that you register your application with NBP under the new flagship name while you handle the `ATTransChangeNameAskTask` transition. However, you should not remove the old NBP registration until you are certain that other applications have not denied the request to change the flagship name. If another application denies the name-change request, the LAP Manager will send an `ATTransCancelNameChange` transition to cancel the name-change request.

Return 0 in the D0 register to indicate that you accept the request to change the flagship name. To deny the request, return a nonzero number in the D0 register.

Cancel-Flagship-Name-Change Transition

When any routine in the AppleTalk Transition Queue refuses a request to change the flagship name, the LAP Manager will send an `ATTransCancelNameChange` transition to any transition event handler routines that acknowledged the `ATTransNameChangeAskTask` transition.

When the LAP Manager calls your transition event handler routine, the stack looks like this:



If your routine registered any entities with NBP under the new flagship name while it processed the `ATTransNameChangeAskTask`, it should remove those entries now. You can make synchronous calls to NBP to remove registration of the entities.

Return a result of 0 in the D0 register to indicate that your routine executed with no errors.

Cable-Range-Change Transition

A cable range is a range of network numbers beginning with the lowest network number and ending with the highest network number defined by a seed router for a network. All node addresses, including multinode addresses, that a system on a network acquires must have a network number within the defined cable range. (For information on multinodes, see the chapter "Multinode Architecture" in this book.)

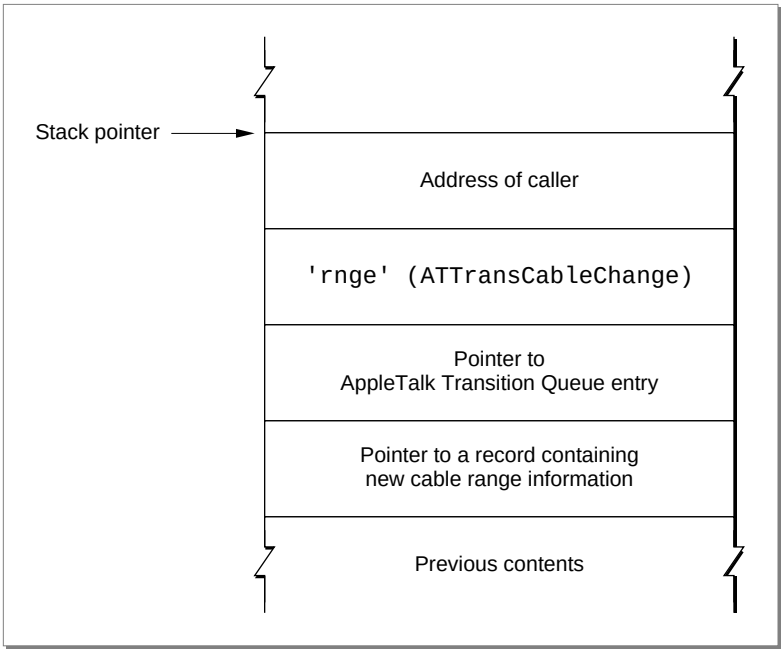
Note
For nonextended networks, the lowest and the highest numbers are the same. ◆

When the cable range of a network changes because, for example, a router on the network shuts down, the LAP Manager will call your transition event handler routine with an `ATTransCableChange` transition. This transition notifies you that the cable range has changed for the network to which your node is connected.

Applications that use multinodes are examples of processes that should handle this transition. For multinode applications, after receiving notification of the cable range change, you should check the new cable range and determine if all the multinode IDs that the application acquired before the transition event occurred are still valid. If you discover multinode IDs that are no longer valid, you should call the `RemoveNode` function to remove them. Then you can call the `AddNode` function to obtain new multinode IDs that are within the valid cable range. See the chapter "Multinode Architecture" for information on `RemoveNode` and `AddNode`.

The LAP Manager sends you notice of a change in the cable range when the following events occur: AppleTalk first identifies the network router, the last router ages out, or AppleTalk first receives a Routing Table Maintenance Protocol (RTMP) broadcast packet that is different from the current range. The `ATTransCableChange` transition is implemented beginning with AppleTalk version 57. This transition event is issued at system task time only.

When the LAP Manager calls your transition event handler routine, the stack looks like this:



The last item on the stack contains a pointer to a record that holds the new high and low cable numbers that identify the cable range. To access this information, you must declare a record of type `TNewCRTrans`. Here is the `TNewCRTrans` record type declaration:

```
TNewCRTrans =
   RECORD
      newCableLo:    Integer; {new low cable in the range, }
                              { received from RTMP}
      newCableHigh:  Integer; {new high cable in the range, }
                              { received from RTMP}
   END;
```

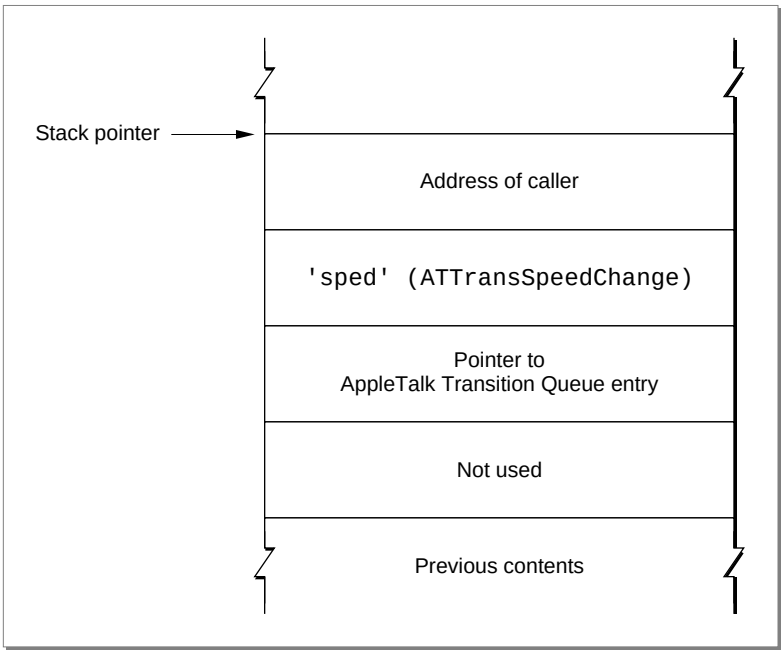
CPU-Speed-Change Transition

Some applications change the CPU speed without rebooting the system. For example, an application may alter the cache states on the 68030 or 68040 CPUs or a third-party accelerator card may support dynamic speed changes made through a control panel `'cdev'` file. Time-dependent processes need to be notified of changes to the CPU speed when these changes occur. If your application changes the CPU speed, you should use the `ATEvent` procedure to send notification of an `ATTransSpeedChange` transition to

time-dependent processes. You must issue this transition event at system task time only. When you call the `ATEvent` procedure, pass `ATTransSpeedChange` as the value of the `event` parameter.

You must always notify LocalTalk when a CPU speed change occurs. LocalTalk includes a module that is time-dependent; the low-level timer values used in this code must be recalculated when the CPU speed changes. Altering the cache state on the 68030 does not affect LocalTalk, whereas altering the cache state on the 68040 does affect the LocalTalk timers. Therefore, an application that dynamically toggles caching on the 68040 should send notification of an `ATTransSpeedChange` transition. If the application does not do this and LocalTalk is the current network connection, the connection will be broken. LocalTalk implemented in AppleTalk version 57 or later recognizes the CPU-speed-change transition event notification.

The transition event handler routine of any time-dependent process should handle the `ATTransSpeedChange` transition notification. When the LAP Manager calls your transition event handler routine, the stack looks like this:



Developer-Defined Transitions

Any AppleTalk transition event code that begins with an uppercase letter (that is, any value in the range $41 00 00 00 through $5A FF FF FF) indicates a developer-defined event. Because you cannot tell how the originator of such an event might interpret a nonzero function result, you must always return 0 in the D0 register for any AppleTalk transition event code that you do not recognize.

When you return a nonzero result code for certain developer-defined transitions, the LAP Manager may call your transition event handler routine a second time with a cancel transition analogous to the cancel-close transition.

Defining Your Own AppleTalk Transition

You can define AppleTalk transitions and use such events to send messages to your own entries in the AppleTalk Transition Queue, or you can define events and make them public for others to use.

You can define your own AppleTalk transition to have any meaning you choose. For example, you might want to call every routine in the AppleTalk Transition Queue each time you open or close a custom protocol stack.

You can use either the `ATEvent` procedure or the `ATPreFlightEvent` function to notify all of the routines in the AppleTalk Transition Queue that your AppleTalk transition has occurred. Whereas the `ATEvent` procedure only calls the routines in the queue with a transition event, the `ATPreFlightEvent` function also allows each routine in the AppleTalk Transition Queue to return a result code and other information to your calling routine.

A developer-defined event, as with any event, always begins with an uppercase letter (that is, any value in the range $41 00 00 00 through $5A FF FF FF).

Note
You can call the `ATEvent` and `ATPreFlightEvent` routines only at virtual-memory safe time. See *Inside Macintosh: Memory* for information on virtual memory. ◆

The LAP Manager and 802.2 Protocol Packets

The Institute of Electrical and Electronics Engineers (IEEE) has defined a series of communications protocols for use on a variety of networks. At the physical level, these protocols include the 802.3 CSMA/CD protocol, the 802.4 token bus protocol, and the 802.5 token ring protocol. At the data-link level, you access these protocols through the IEEE 802.2 Logical Link Control (LLC) protocol. If you write an application that handles 802.2 Type 1 data packets, you must include a protocol handler to read the data. You can install your application as a client of the LAP Manager to receive 802.2 packets from an Ethernet, token ring, or FDDI driver.

The LAP Manager includes two routines that allow you to attach and detach protocol handlers for 802.2 Type 1 data packets: the `L802Attach` and `L802Detach` routines. The LAP Manager contains a generic protocol handler that receives data from the hardware device drivers and determines for which application the 802.2 packet is meant based on the protocol type. The LAP Manager's protocol handler then calls the destination application's protocol handler to read in the data. This section uses Ethernet to illustrate how this process works; however, the same process applies to token ring and FDDI packets.

The ANSI/IEEE standards for the 802 protocols are published by the IEEE. The first 14 bytes of a packet sent or received by the .ENET driver constitute the header. The first 12 bytes consist of the destination and source data-link addresses, such as the Ethernet hardware addresses. If the value of the last 2 bytes in the header is greater than 1500, then the .ENET driver treats that field as an Ethernet protocol type discriminator; this

indicates that the packet is an Ethernet Phase 1 packet. If the value of the last 2 bytes in the header is less than or equal to 1500, then the field contains the length of the 802.2 packet, not including the 14-byte header, and this indicates that the packet is an Ethernet Phase 2 packet. The .ENET driver passes all Phase 2 packets to the LAP Manager.

The IEEE LLC standard defines the concept of a Service Access Point (SAP). A SAP is a 1-byte value that is used to distinguish the different protocols using 802.2 in a single node. Most SAPs are reserved for use by IEEE standard protocols. IEEE has reserved one SAP, whose value is $AA, for use by protocols other than the standard IEEE protocols. AppleTalk and many other protocol families use SAP $AA. Because other protocol families can also use this SAP, the value of another field that contains the ***subnetwork access protocol (SNAP)*** type is used to discriminate for which protocol family a packet with a destination subnetwork access protocol value of $AA is intended.

At the physical level, a packet contains the 802.3 header, the data field of which contains either an Ethernet protocol type discriminator (for Phase 1 packets) or the 802.2 packet length (for Phase 2 packets). For all Phase 2 packets, the LAP Manager receives the entire 802.3 packet from the .ENET driver. The first 14 bytes of the 802.3 data constitute the frame header, and they are followed by the 802.2 protocol header.

The first byte of the 802.2 header is the ***destination service access point (DSAP).*** If the DSAP value is equal to $AA, then the first 5 bytes of the 802.2 data constitute a SNAP protocol type discriminator. If the SNAP type value is $00000080F3, indicating the AppleTalk Address Resolution Protocol (AARP), then the next 4 bytes of the 802.2 data constitute the AARP packet type field. AARP is not discussed at length in this book; for complete information about AARP, see *Inside AppleTalk,* second edition.

Figure 10-2 shows an Ethernet packet containing AppleTalk Phase 1 data. Phase 1 packets are the original version of Ethernet packets. The last 2 bytes in the header contain a value greater than 1500, indicating that this field is to be treated as a protocol type discriminator.

Figure 10-2 Ethernet Phase 1 packet formats

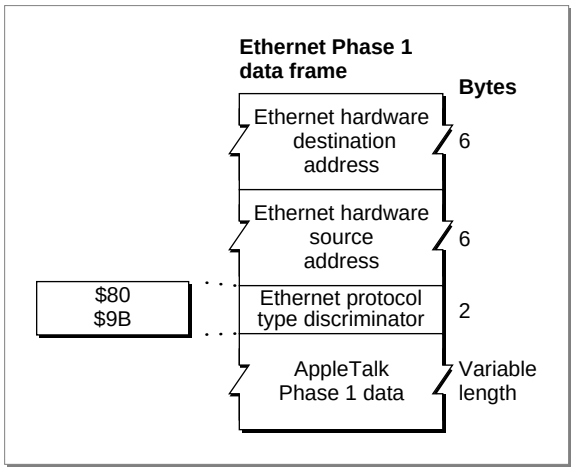
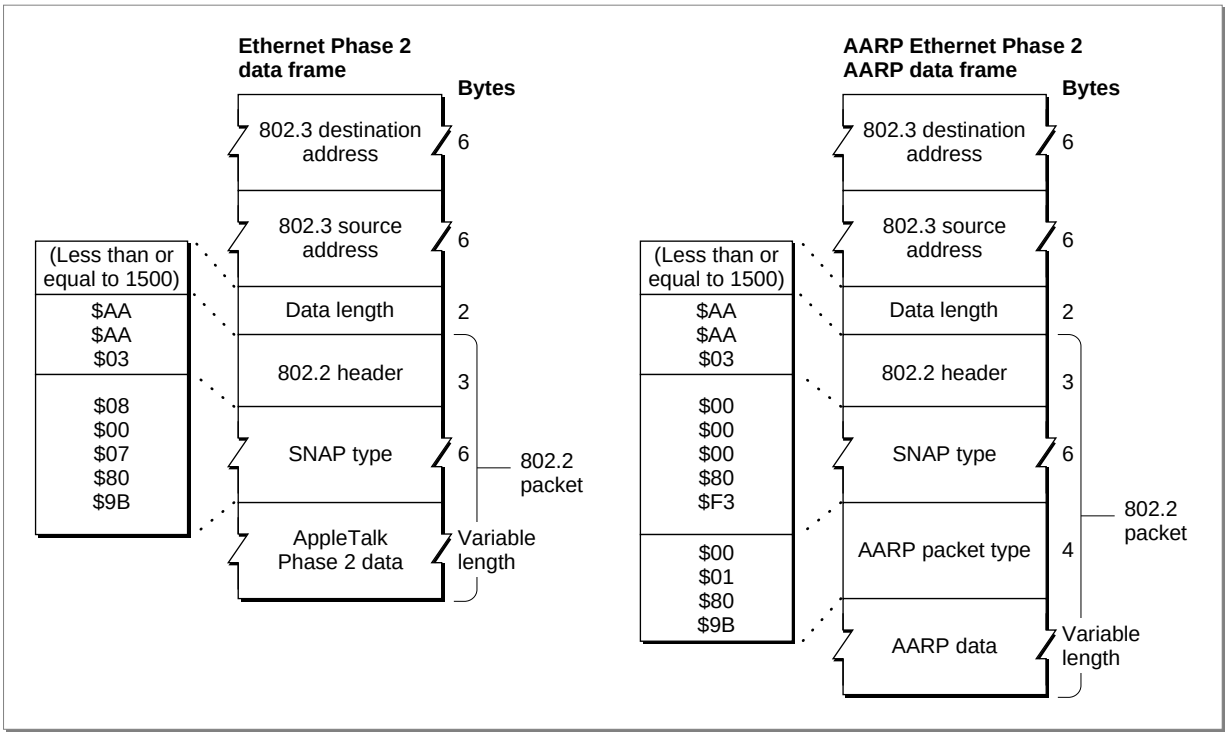


Figure 10-3 shows two Phase 2 packets. For Phase 2 packets, the last 2 bytes of the 802.3 header contain the 802.2 packet length, not including the 14-byte header; the 802.2 packet length is a value from 0 through 1500.

The data frame on the left shows an Ethernet 802.3 packet containing an 802.2 packet that holds AppleTalk Phase 2 data. The Ethernet driver would deliver this entire packet to the LAP Manager; the 802.2 packet is enclosed in the 802.3 packet, which is also referred to as a frame. The data frame on the right shows an Ethernet 802.3 packet containing an 802.2 packet to be delivered to the Phase 2 Ethernet AARP handler; the SNAP type value is $00000080F3, indicating the AppleTalk Address Resolution Protocol (AARP).

Figure 10-3 Ethernet Phase 2 packet formats



When you call the `L802Attach` routine, you provide a pointer to your protocol handler, the reference number of the .ENET driver, and a pointer to a string containing one or more type fields. The type fields indicate the DSAP value and any other protocol type fields (such as the SNAP type and the AARP type). The LAP Manager delivers to your protocol handler any 802.2 data packets that have the protocol type you specify.

Attaching and Detaching 802.2 Protocol Handlers

You must use the LAP Manager to attach your protocol handler for 802.2 protocols to receive Ethernet Phase 2 packets and all token ring and FDDI packets.

The LAP Manager is designed to install a generic protocol handler that receives packets from the hardware device drivers for 802.2 protocols and that also serves as a dispatcher. The LAP Manager's protocol handler maintains an index of registered protocol types and pointers to their protocol handlers. When an application calls the LAP Manager to attach a protocol handler, the LAP Manager adds an entry for the application's protocol type and protocol handler to its protocol handler index.

The LAP Manager's protocol handler determines for which application data is meant. When processing a packet, the LAP Manager reads the destination SAP; if the SAP value is $AA, the LAP Manager then checks the SNAP header for the protocol type, and then it searches for a protocol type match in its protocol handler index. If the LAP Manager finds a protocol type match, it calls the destination application's protocol handler to read in the data. You cannot replace or override the permanent LAP Manager protocol handler.

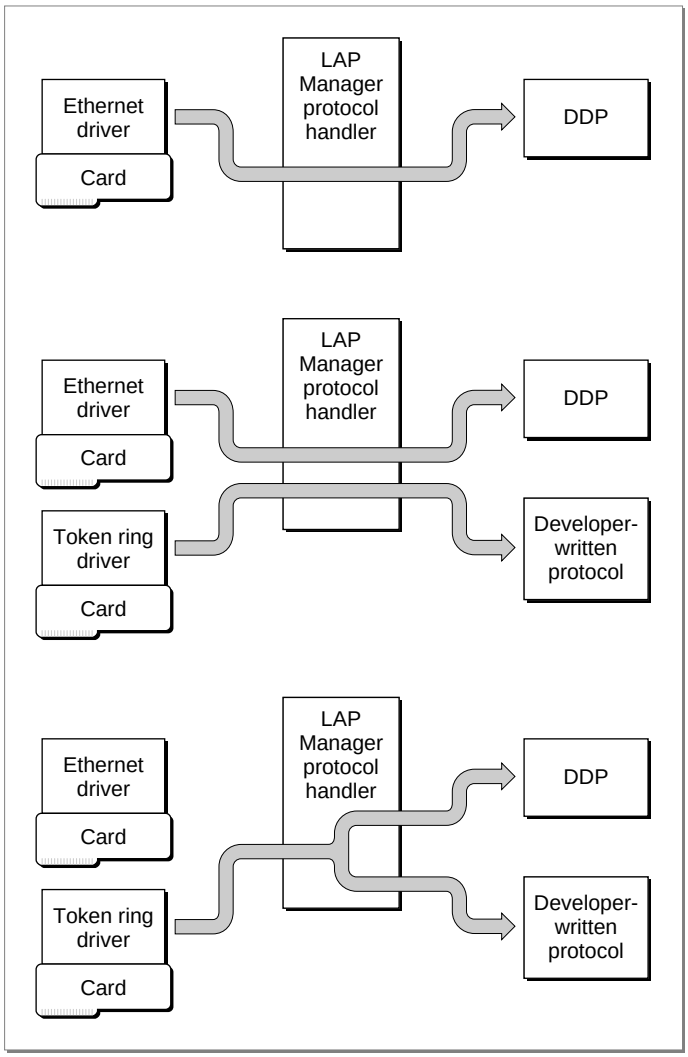
The first time that a process or application calls the LAP Manager to attach a protocol handler for 802.2 packets, the LAP Manager calls the specified hardware device driver directly to install its own generic protocol handler. The LAP Manager then registers in its index the protocol handler and the protocol type for the process that initially called it. When a process or application subsequently calls the LAP Manager to attach a protocol handler to receive 802.2 packets from the same type of hardware device driver, the LAP Manager simply adds the protocol handler and protocol type information for that process to its index.

The LAP Manager allows for the concurrent use of hardware device drivers by more than one application. For example, Figure 10-4 shows three scenarios. In the first instance at the top of the figure, only AppleTalk is using the Ethernet driver to receive data; AppleTalk always uses the LAP Manager, which provides for its link independence.

In the second instance in the middle of the figure, both AppleTalk and a developer-written application have attached their protocol handlers to the LAP Manager. AppleTalk is configured to use the Ethernet driver; when the LAP Manager's protocol handler reads a packet, it determines if the data is meant for AppleTalk, and if so, the LAP Manager calls the DDP protocol handler to receive the data. If the data is meant for the other application, the LAP Manager calls that application's protocol handler.

In the third instance at the bottom of the figure, both AppleTalk and the developer-written application have attached their protocol handlers to the LAP Manager to receive data from the token ring driver. The LAP Manager receives the data, determines the destination, then calls the appropriate protocol handler, either the DDP protocol handler or the developer-written application's protocol handler to receive the data.

Figure 10-4 Using the LAP Manager to receive data for 802.2 protocols



There are no high-level interfaces for the LAP Manager 802.2 protocol routines. You call these routines from assembly language by placing a routine selector in the D0 register and executing a JSR instruction to an offset 2 bytes past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18).

Before you call these routines, you must place the reference number of the .ENET driver in the D2 register and a pointer to the protocol type specification in the A1 register. Before you call the `L802Attach` routine, you must also place a pointer to your protocol handler in the A0 register. Both routines return a nonzero value in the D0 register if there is an error.

Listing 10-7 shows how to call either the LAP Manager's `L802Attach` or `L802Detach` routine from assembly language. To specify either of these routines, you place the routine selector in register D0, as indicated in the sample code.

Listing 10-7 Calling a LAP Manager 802.2 routine from assembly language

```
LAPMgrPtr   EQU     $B18            ;entry point for LAP Manager
LAPMgrCall  EQU     2               ;offset to LAP Manager
                                    ; routines
L802Entry   EQU     *               ;802 routine entry

            MOVEQ   #RSel,D0        ;place the routine selector
                                    ; in D0
            MOVEQ   #refNum,D2      ;place the driver reference
                                    ; number in D2
            MOVE.L  PHndlrPtr,A0    ;put pointer to protocol
                                    ; handler in A0 (L802Attach
                                    ; only)
            MOVE.L  PSpecPtr,A1     ;put pointer to protocol
                                    ; specification in A1
            MOVE.L  LAPMgrPtr,An    ;put pointer to LAP Mgr in An
            JSR     LAPMgrCall(An)  ;jump to start of LAP Mgr
                                    ; routines
```

For information on the protocol type specification whose pointer you place in register A1, see "L802Attach" beginning on page 10-40.

LAP Manager Reference

This section describes the data structures and routines that are specific to the LAP Manager.

The "Data Structures" section shows the Pascal data structure for the AppleTalk Transition Queue entry record.

The "Routines" section describes routines for adding and removing an AppleTalk Transition Queue entry, requesting permission to close the .MPP driver, notifying the routines specified by AppleTalk Transition Queue entries when a transition occurs that your application has defined, and attaching and detaching your own 802.2 protocol handler for Type 1 packets.

Data Structures

This section describes the `ATQEntry` record that you use to specify your AppleTalk Transition Queue entry routine to be called when a transition event occurs. You pass a pointer to this record as a parameter to the `LAPAddATQ` function, which you call to place your entry in the AppleTalk Transition Queue.

The AppleTalk Transition Queue Entry

You use the AppleTalk Transition Queue entry record to specify an entry to be added to the transition queue. The `ATQEntry` data type defines an AppleTalk Transition Queue entry.

```
TYPE ATQEntry =
RECORD
   qLink:      ATQEntryPtr;      {next queue entry}
   qType:      Integer;          {reserved}
   CallAddr:   ProcPtr;          {pointer to your routine}
END;
```

Field descriptions

`qLink`	A pointer to the next queue entry. Set this field to `NIL`; the LAP Manager fills it in when an application adds another entry to the queue.
`qType`	Reserved.
`CallAddr`	A pointer to a transition event handler routine that you provide. The LAP Manager calls your routine when an AppleTalk transition event occurs.

Because you provide the memory for the AppleTalk Transition Queue entry, you can add as many fields to the end of the entry as you wish for your own purposes. Whenever your routine is called, the caller provides you with a pointer to the queue entry so that you can have access to the information you stored at the end of your queue entry.

Routines

This section describes the LAP Manager's Pascal interface to the AppleTalk Transition Queue that allows you to place an entry for your application in the queue so that you will be notified when an AppleTalk transition occurs.

The Pascal interface to the AppleTalk Transition Queue consists of four routines:

- The `LAPAddATQ` function adds an entry to the AppleTalk Transition Queue.
- The `LAPRmvATQ` function removes an entry from the AppleTalk Transition Queue.
- The `ATEvent` procedure calls all the entries in the AppleTalk Transition Queue with an AppleTalk transition event code that you specify.

- The `ATPreFlightEvent` function calls all the entries in the AppleTalk Transition Queue with an AppleTalk transition event code that you specify in the `event` parameter. If any routine returns a nonzero function result, the LAP Manager calls all of the entries with the transition event code that you specify in `ATPreFlightEvent` function's `cancel` parameter.

This section also describes the LAP Manager's assembly-language interface that allows you to install and remove your own protocol handler for a specific IEEE 802.2 protocol type. You can write a protocol handler application that reads 802.2 Type 1 data packets, and you can install your application as a client of the LAP Manager.

The assembly-language routines that allow you to attach and detach protocol handlers for 802.2 Type 1 data packets are

- the `L802Attach` routine, which installs your protocol handler for a specific IEEE 802.2 protocol type
- the `L802Detach` routine, which detaches from the LAP Manager your protocol handler for a specific IEEE 802.2 protocol type

Note
The ANSI/IEEE standards for the 802 protocols are published by the IEEE. ◆

Adding and Removing AppleTalk Transition Queue Entries

This section describes the `LAPAddATQ` function that you use to add an entry to the AppleTalk Transition Queue and the `LAPRmvATQ` function that you use to remove an entry from the queue.

LAPAddATQ

The `LAPAddATQ` function adds an entry to the AppleTalk Transition Queue.

```
FUNCTION LAPAddATQ (theATQEntry: ATQEntryPtr): OSErr;
```

`theATQEntry`
A pointer to a record of type `ATQEntry` to be added to the AppleTalk Transition Queue.

DESCRIPTION

You use the `LAPAddATQ` function to add an entry for your application to the AppleTalk Transition Queue. Before you call the `LAPAddATQ` function, you must create an AppleTalk Transition Queue entry record of type `ATQEntry` that defines your entry. "The AppleTalk Transition Queue Entry" on page 10-33 describes the `ATQEntry` record. You provide a pointer to this record as the value of the `theATQEntry` parameter when you call the `LAPAddATQ` function.

In the `CallAddr` field of the AppleTalk Transition Queue entry record, you provide a pointer to a routine that the LAP Manager is to call when an AppleTalk transition event occurs. The LAP Manager calls your routine to notify you when any of the following events occurs:

- A process opens the .MPP driver.
- A process requests permission to close AppleTalk.
- A process closes the .MPP driver.
- A request to close AppleTalk is canceled. One of the routines pointed to by an entry in the AppleTalk Transition Queue denies permission to close AppleTalk, and so the request to do so is canceled.
- A process calls the `ATEvent` procedure or the `ATPreFlightEvent` function to send its own AppleTalk transition event to the entries in the AppleTalk Transition Queue.
- A process reports that it is changing the flagship name.
- A process makes a request to change the flagship name.
- A request to change the flagship name is canceled. One process denies another's request to change the flagship name, and so the request is canceled.
- The network connectivity has changed. This transition event is sent if a node is connected to an AppleTalk network and, for some reason, a particular interconnected AppleTalk network is longer be reachable.
- The cable range for the current network has been changed.
- The speed of the CPU has been changed.

SPECIAL CONSIDERATIONS

You must allocate nonrelocatable memory for the `ATQEntry` record and not alter or manipulate this memory until you remove the AppleTalk Transition Queue entry from the transition queue using the `LAPRmvATQ` function.

When LAP Manager calls your transition event handler routine, the LAP Manager passes parameters to your routine using the C stack calling conventions, and expects your routine to return a result in register D0. If you write your transition event handler routine in Pascal, you must use an assembly glue code routine. For a sample glue code routine, see "Writing a Transition Event Handler Routine Using Pascal" beginning on page 10-11.

ASSEMBLY-LANGUAGE INFORMATION

From assembly language, you add an AppleTalk Transition Queue entry by placing a routine selector in the D0 register, placing a pointer to your AppleTalk Transition Queue entry in the A0 register, and executing a JSR instruction to an offset past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18). The offset to the LAP Manager routines is given by the constant `LAPMgrCall` (2).

Registers on entry

D0 23

A0 Pointer to AppleTalk Transition Queue entry

Registers on exit

D0 Result code

RESULT CODES

`noErr`	0	No error

SEE ALSO

"Adding an Entry to the AppleTalk Transition Queue" on page 10-7 describes the process of creating an AppleTalk Transition Queue entry and adding it to the queue.

For the details of each transition, see "How the LAP Manager Calls Your Transition Event Handler Routine" beginning on page 10-9.

LAPRmvATQ

The `LAPRmvATQ` function removes an entry from the AppleTalk Transition Queue.

```
FUNCTION LAPRmvATQ (theATQEntry: ATQEntryPtr): OSErr;
```

`theATQEntry`
A pointer to the `ATQEntry` record to be removed from the AppleTalk Transition Queue.

DESCRIPTION

You use the `LAPRmvATQ` function to remove your application's entry from the AppleTalk Transition Queue. To identify the entry to be removed, you pass the `LAPRmvATQ` function the same pointer to the AppleTalk Transition Queue entry record that you provided as the value of the `theATQEntry` parameter when you called the `LAPAddATQ` function to place the entry in the queue.

SPECIAL CONSIDERATIONS

You must not call the `LAPRmvATQ` function at interrupt time or through a callback routine. This restriction is to prevent any routine from removing an entry from the AppleTalk Transition Queue while another routine is in the process of adding or removing an entry.

ASSEMBLY-LANGUAGE INFORMATION

From assembly language, you remove an AppleTalk Transition Queue entry by placing a routine selector in the D0 register, placing a pointer to your AppleTalk Transition Queue entry in the A0 register, and executing a JSR instruction to an offset past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18). The offset to the LAP Manager routines is given by the constant `LAPMgrCall` (2).

Registers on entry

D0 24

A0 Pointer to AppleTalk Transition Queue entry

Registers on exit

D0 Result code

RESULT CODES

`noErr`	0	No error
`qErr`	–1	Queue element not found

Notifying Routines When Your Application-Defined Transition Occurs

This section describes the `ATEvent` and `ATPreFlightEvent` routines that you can use to notify all of the entries in the AppleTalk Transition Queue that an AppleTalk transition that you have defined has occurred.

You can define your own AppleTalk transition to have any meaning you choose. For example, you might want to call every routine in the AppleTalk Transition Queue each time you open an AppleTalk Data Stream Protocol (ADSP) connection.

ATEvent

The `ATEvent` procedure calls the routines specified by each of the entries in the AppleTalk Transition Queue with notification of a transition event that you have defined.

```
PROCEDURE ATEvent (event: LongInt; infoPtr: Ptr);
```

`event` The AppleTalk transition event code for your application-defined transition. This can be any four-character string that starts with an uppercase letter—that is, any value in the range $41 00 00 00 through $5A FF FF FF.

`infoPtr` A pointer to information that you make available to the AppleTalk Transition Queue entry routines. If you do not want to pass any information to these routines, set the `infoPtr` parameter to `NIL`.

DESCRIPTION

The `ATEvent` procedure calls the routines in the queue with the AppleTalk transition event code you specify in the `event` parameter. You can use the `infoPtr` parameter to point to any information that you want to make available to the transition event handler routines; for an ADSP-open transition, for example, you might pass a pointer to the parameter block used by the `dspOpen` routine.

You use the `ATEvent` procedure to send notification of an `ATTransSpeedChange` transition to time-dependent processes. You must send this transition event notification if your application changes the CPU speed. Note that you must issue this transition event at system task time only.

For transition events that you define, you can issue the `ATEvent` procedure at interrupt time provided that the transition event handler routines follow the standard rules for interrupt operation.

SPECIAL CONSIDERATIONS

You can call the `ATEvent` procedure only at virtual-memory safe time.

AppleTalk transitions defined by developers might return other result codes.

RESULT CODES

`noErr`	0	No error, or unrecognized event code

SEE ALSO

For more information about the `ATTransSpeedChange` event, see "CPU-Speed-Change Transition" on page 10-25.

For more information about developer-defined transition events, see "Developer-Defined Transitions" on page 10-26 and "Defining Your Own AppleTalk Transition" on page 10-27.

For information on virtual memory, see *Inside Macintosh: Memory*.

ATPreFlightEvent

The `ATPreFlightEvent` function calls the routines specified by each of the entries in the AppleTalk Transition Queue with notification of a transition event that you have defined and allows each routine in the AppleTalk Transition Queue to return a result code and other information to your calling routine.

```
FUNCTION ATPreFlightEvent (event,cancel: LongInt;
                           infoPtr: Ptr): OSErr;
```

`event`	The AppleTalk transition event code for the initial transition about which you want to notify the AppleTalk Transition Queue event routines. This code can be any four-character string that starts with an uppercase letter—that is, any value in the range $41 00 00 00 through $5A FF FF FF.
`cancel`	The AppleTalk transition event code for the transition that notifies the AppleTalk Transition Queue event routines that your original transition notification is canceled. This code can be any four-character string that starts with an uppercase letter—that is, any value in the range $41 00 00 00 through $5A FF FF FF.
`infoPtr`	A pointer to information that you make available to the AppleTalk Transition Queue entry routines. If you do not want to pass any information to these routines, set the `infoPtr` parameter to `NIL`.

DESCRIPTION

The `ATPreFlightEvent` function calls all of the routines in the AppleTalk Transition Queue with the AppleTalk transition event code you specify in the `event` parameter. If any routine in the AppleTalk Transition Queue returns a nonzero function result, the `ATPreFlightEvent` function calls each of the routines that it has already called, this time with the AppleTalk transition event code you specify in the `cancel` parameter.

SPECIAL CONSIDERATIONS

You can call the `ATPreFlightEvent` function only at virtual-memory safe time.

AppleTalk transitions defined by developers might return other result codes.

RESULT CODES

`noErr`	0	No error, or unrecognized event code

SEE ALSO

See *Inside Macintosh: Memory* for information on virtual memory.

For information about developer-defined transition events, see "Developer-Defined Transitions" on page 10-26 and "Defining Your Own AppleTalk Transition" on page 10-27.

Attaching and Detaching 802.2 Protocol Handlers

You can attach to the LAP Manager your own protocol handler for 802.2 protocols. The LAP Manager has a generic protocol handler that it attaches at the hardware device driver level for all 802.2 packets; you must not replace or override this protocol handler. You can also detach from the LAP Manager any 802.2 protocol handler that you have provided and attached.

You use the `L802Attach` routine to attach your protocol handler and the `L802Detach` routine to detach your protocol handler. There are no high-level interfaces for the LAP Manager 802.2 protocol routines. You must call these routines from assembly language.

L802Attach

The `L802Attach` routine attaches to the LAP Manager a protocol handler for a specific IEEE 802.2 protocol type.

DESCRIPTION

You call the `L802Attach` routine from assembly language by placing the routine selector of 21 in the D0 register and the reference number of the Ethernet, token ring, or FDDI driver in the D2 register that the `OpenSlot` or `OpenDriver` function returns. Then, you execute a JSR instruction to an offset 2 bytes past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18).

Here are the register contents that you supply on entry and the value that is returned to you.

Registers on entry

D0 21

D2 Reference number of hardware device driver

A0 Pointer to your protocol handler

A1 Pointer to protocol-type specification

Registers on exit

D0 Nonzero if error

You must put a pointer to your protocol handler in the A0 register and a pointer to the protocol-type specification for this protocol handler in the A1 register. The protocol-type specification consists of one or more protocol-type fields, each preceded by a length byte. The LAP Manager reads the fields in the 802.2 data packet header to determine to which protocol handler (if any) to deliver the packet. The first type field in your protocol specification is the 1-byte DSAP. If the DSAP type field is equal to `$AA`, then the packet is a SNAP packet. In this case, the protocol-type specification must contain a second type field, the 5-byte SNAP type. If the SNAP type field is `$00000080F3`, indicating the AppleTalk Address Resolution Protocol (AARP), then the protocol-type specification must contain a third type field, the 4-byte AARP protocol type. Terminate the list of protocol-type fields with a byte of zeros.

The following protocol-type specification, for example, is for the permanent LAP Manager protocol handler for an 802.3 packet containing AppleTalk data. The .ENET driver would deliver this packet to the LAP Manager. The first byte, `$01`, is the length byte for the first protocol-type field (the DSAP type field), `$AA`, contained in the second byte. The DSAP value of `$AA` is reserved for use with protocol-type specifications that include a SNAP field. The third byte, `$05`, is the length byte for the next protocol-type field, the SNAP type field, `$0800078098`. The SNAP value of `$08 00 07 80 9B` is reserved for AppleTalk data. The final byte (`$00`) terminates the type specification.

```
01 AA 05 08 00 07 80 9B 00
```

The following protocol-type specification is for the permanent LAP Manager protocol handler for an 802.3 packet to be delivered to the EtherTalk AARP handler. Notice that the SNAP field is followed by an additional type field, the AARP protocol type.

```
01 AA 05 00 00 00 80 F3 04 00 01 80 9B 00
```

The SNAP value of `$00 00 00 80 F3` is reserved for AARP data. The AARP protocol type value of `$00 01 80 9B` is reserved for Ethernet AARP packets.

SPECIAL CONSIDERATIONS

For token ring, the Apple Computer, Inc. specification for the device driver that the hardware vendor must implement requires that the driver process only SNAP packets, that is, packets with a SAP value of $AA. For Ethernet and FDDI, your protocol can receive packets with a SAP value of $AA or any other SAP value.

You can only use the `L802Attach` routine if the hardware device driver interface conforms to the Apple specification for that driver type.

RESULT CODES

The `L802Attach` routine returns a nonzero value in the D0 register if there is an error.

SEE ALSO

See the "The LAP Manager and 802.2 Protocol Packets" on page 10-27 and the ANSI/IEEE standard 802.2 for more information about 802.2 protocols, and see *Inside AppleTalk,* second edition, for more information about AARP.

See *Inside Macintosh: Devices* for information on the `OpenSlot` function.

L802Detach

The `L802Detach` routine detaches from the LAP Manager a protocol handler for a specific IEEE 802.2 protocol type.

DESCRIPTION

You use the `L802Detach` routine to remove a protocol handler that you have written and attached using the `L802Attach` routine. You call the `L802Detach` routine from assembly language by placing the routine selector of 22 in the D0 register and the reference number of the Ethernet, token ring, or FDDI driver in the D2 register that the `OpenSlot` or `OpenDriver` function returns. Then, you execute a JSR instruction to an offset 2 bytes past the start of the LAP Manager. The start of the LAP Manager is contained in the global variable `LAPMgrPtr` ($B18).

Here are the register contents that you supply on entry and the value that is returned to you.

Registers on entry

D0 22

D2 Reference number of the hardware device driver

A1 Pointer to protocol specification

Registers on exit

D0 Nonzero if error

You must put a pointer to the protocol-type specification for this protocol handler in the A1 register. You must specify exactly the same protocol type as you specified for the `L802Attach` routine when you attached the protocol handler.

RESULT CODES

`L802Detach` routine returns a nonzero value in the D0 register if there is an error.

SEE ALSO

See *Inside Macintosh: Devices* for information on the `OpenSlot` and `OpenDriver` functions.

Summary of the LAP Manager

Pascal Summary

Constants

```
CONST
   {Transition Queue transition types}
   ATTransOpen          =       0;       {AppleTalk has been opened}
   ATTransClose         =       2;       {AppleTalk is about to close}
   ATTransClosePrep     =       3;       {permission to close AppleTalk}
   ATTransCancelClose   =       4;       {cancel the ClosePrep transition}

{To use the following six constants, you must first declare them in your }
{ application. They are not included in the MPW interface files.}
   ATTransNetworkTransition   =  5;       {change in network connection for }
                                          { Apple Remote Access (ARA)}
   ATTransNameChangeTellTask  =  6;       {flagship name change}
   ATTransNameChangeAskTask   =  7;       {permission to change flagship }
                                          { name}
   ATTransCancelNameChange    =  8;       {cancel flagship name change}
   ATTransCableChange         =  'rnge';  {change in cable range}
   ATTransSpeedChange         =  'sped';  {change in CPU speed}
```

Data Types

AppleTalk Transition Queue Entry

```
TYPE ATQEntry =
RECORD
   qLink:      ATQEntryPtr;      {next queue entry}
   qType:      Integer;          {reserved}
   CallAddr:   ProcPtr;          {pointer to your routine}
END;

ATQEntryPtr = ^ATQEntry;
```

Routines

Adding and Removing AppleTalk Transition Queue Entries

```
FUNCTION LAPAddATQ          (theATQEntry: ATQEntryPtr): OSErr;
FUNCTION LAPRmvATQ          (theATQEntry: ATQEntryPtr): OSErr;
```

Notifying Routines When Your Application-Defined Transition Occurs

```
PROCEDURE ATEvent           (event: LongInt; infoPtr: Ptr);
FUNCTION ATPreFlightEvent   (event: LongInt; cancel: LongInt; infoPtr: Ptr):
                             OSErr;
```

C Summary

Constants

```
/*LAP Manager parameter constants*/
#define LAPprotType LAP.protType
#define LAPwdsPointer LAP.LAPptrs.wdsPointer
#define LAPhandler LAP.LAPptrs.handler

enum {                                  /*AppleTalk Transition Queue */
                                        /* transition types*/
ATTransOpen                 =     0,    /*AppleTalk has opened*/
ATTransClose                =     2,    /*AppleTalk is about to close*/
ATTransClosePrep            =     3,    /*permission to close AppleTalk*/
ATTransCancelClose          =     4,    /*cancel ClosePrep transition*/

/*To use the following six constants, you must first define them in */
/* your application. They are not defined in the MPW interface files.*/
ATTransNetworkTransition    =     5,    /*change in network connection */
                                        /* for ARA*/
ATTransNameChangeTellTask   =     6,    /*flagship name change*/
ATTransNameChangeAskTask    =     7,    /*permission to change */
                                        /* flagship name*/
ATTransCancelNameChange     =     8,    /*cancel flagship name change*/
ATTransCableChange          =     'rnge',   /*change in cable range*/
ATTransSpeedChange          =     'sped',   /*change in CPU speed*/
};
```

Data Types

AppleTalk Transition Queue Entry

```
struct ATQEntry {
   struct ATQEntry   *qLink;        /*reserved*/
   short             qType;         /*reserved*/
   ProcPtr           CallAddr;      /*pointer to your routine*/
};

typedef struct ATQEntry ATQEntry;
typedef ATQEntry *ATQEntryPtr;
```

Routines

Adding and Removing AppleTalk Transition Queue Entries

```
pascal OSErr                LAPAddATQ(ATQEntryPtr theATQEntry);
pascal OSErr                LAPRmvATQ(ATQEntryPtr theATQEntry);
```

Notifying Routines When Your Application-Defined Transition Occurs

```
pascal void                 ATEvent(long event, Ptr infoPtr);
pascal OSErr                ATPreFlightEvent(long event, long cancel, Ptr
                             infoPtr);
```

Assembly-Language Summary

Constants

```
;routine selectors to attach and detach an 802.2 protocol handler
L802Attach          EQU    21         ;attach an 802.2 protocol handler
L802Detach          EQU    22         ;detach an 802.2 protocol handler

;miscellaneous LAP Manager values
LAPMgrPtr           EQU   $B18        ;entry point for LAP Manager
LAPMgrCall          EQU   2           ;offset to LAP routines
LAddAEQ             EQU   23          ;LAPAddATQ routine selector
LRmvAEQ             EQU   24          ;LAPRmvATQ routine selector
```

Data Structures

AppleTalk Transition Queue Entry Data Structure

0	`AeQQLink`	long	next queue entry
4	`AeQQType`	word	reserved
6	`AeQCallAddr`	long	pointer to your transition event handler routine

Result Codes

`noErr`	0	No error, or unrecognized event code
`qErr`	–1	Queue element not found