Using ATP Information about the AppleTalk Transaction Protocol

This section documents the alternate interface. It is recommended that you use the preferred interface instead. See **The Preferred Interface** for additional information.

Before you can use ATP on a Macintosh 128K, the .ATP driver must be read from the system resource file via an **ATPLoad** call. the .ATP driver loads itself into the application heap and installs a task into the vertical retrace queue.

Warning: When another application starts up, the application heap is reinitialized; on a Macintosh 128K, this means that the ATP code is lost (and must be reloaded by the next application).

When you're through using ATP on a Macintosh 128K, call **ATPUnload**--the system will be returned to the state it was in before the .ATP driver was opened.

On any Macintosh later than the 128K, the .ATP driver will have been loaded into the system heap either at system startup or upon execution of **MPPOpen** or **ATPLoad**. **ATPUnload** has no effect on a Macintosh later than the 128K.

To send a transaction request, call **ATPSndRequest** or **ATPRequest**. The .ATP driver will automatically select and open a socket through which the request datagram will be sent, and through which the response datagrams will be received. The transaction requester cannot specify the number of this socket. However, the requester must specify the full network address (network number, node ID, and socket number) of the socket to which the request is to be sent. The socket is known as the responding socket, and its address must be known in advance by the requester.

Note: Upon opening,. the ability to send a packet to one's own node (intranode delivery) is disabled. This feature of the **AppleTalk** Manager can be manipulated through the **SetSelfSend** function. Once enabled it is possible, at all levels, to send packets to entities within one's own node. An example of where this might be desirable is in an application sending data to a print spooler that is actually running in the background on the same node.

Enabling or disabling this feature affects the entire node and should be performed with care. For instance, a desk accessory may not expect to receive names from within its own node as a response to an NBP look-up; enabling this feature from an application could break the desk accessory.

At the responder's end, before a transaction request can be received, a responding socket must be opened, and the appropriate calls be made, to receive a request. To do this, the responder first makes an **ATPOpenSocket** call which allows the responder to specify the address (or part of it) of the requesters from whom it's willing to accept transaction requests. Then it issues an **ATPGetRequest** call to provide ATP with a buffer for receiving a request; when a request is received, **ATPGetRequest** is complete. The responder can queue up several **ATPGetRequest** calls, each of which will be completed as requests are received.

Upon receiving a request, the responder performs the requested operation,

and then prepares the information to be returned to the requester. It then calls **ATPSndRsp** (or **ATPResponse**) to send the response. Actually, the responder can issue the **ATPSndRsp** call with only part (or none) of the response specified. Additional portions of the response can be sent later by calling **ATPAddRsp**.

The **ATPSndRsp** and **ATPAddRsp** calls provide flexibility in the design (and range of types) of transaction responders. For instance, the responder may, for some reason, be forced to send the responses out of sequence. Also, there might be memory constraints that force sending the complete transaction response in parts. Even though eight response datagrams might need to be sent, the responder might have only enough memory to build one datagram at a time. In this case, it would build the first response datagram and call **ATPSndRsp** to send it; and so on, for the second through eighth response datagrams.

A responder can close a responding socket by calling **ATPCloseSocket**. This call cancels all pending ATP calls for that socket, such as **ATPGetRequest**, **ATPSndRsp**, **ATPResponse**.

For exactly-once transactions, the **ATPSndRsp** and **ATPAddRsp** calls do not terminate until the entire transaction has completed (that is, the responding end receives a release packet, or the release timer has expired).

To cancel a pending asynchronous **ATPSndRequest** or **ATPRequest** call, call **ATPReqCancel**. To cancel a pending asynchronous **ATPSndRsp** or **ATPResponse** call, call **ATPRspCancel**. Pending asynchronous **ATPGetRequest** calls can be canceled only by issuing the **ATPCloseSocket** call, but that will cancel all outstanding calls for that socket.

Warning: You cannot reuse a variable of type **ATATPRec** passed to an ATP routine until the entire transaction has either been completed or canceled.

```
// The requesting side of an ATP transaction that asks for a 512-byte disk block
// from the responding end.  The block number of the file is a byte and is
// contained in myBuffer[0].
// Assuming inclusion of <MacHeaders>

#include <AppleTalk.h>

void DoError (OSErr myErr);

main ()
{
    ATATPRecHandle   myABRecord;
    BDSPtr           myBDSPtr;
    char        myBuffer[512];
    short       myErr;
    Boolean          async;

    myErr = ATPLoad();
    if (myErr)
        DoError(myErr);
        // Maybe serial port B isn't available for use by AppleTalk

    else {
```

```
        // Prepare the BDS; allocate space for a one-element BDS

        myBDSPtr = (BDSPtr) NewPtr(sizeof(BDSElement));
        myBDSPtr[0]->buffSize = 512; // Size of our buffer used in
                                     // reception
        myBDSPtr[0]->buffPtr = myBuffer;    // Pointer to the buffer

        // Prepare the ATATPRec

        myBuffer[0] = 1;
        myABRecord = (ATATPRecHandle) NewHandle (atpSize);

        (*myABRecord)->atpAddress.aNet = 0;
        (*myABRecord)->atpAddress.aNode = 30;   // We probably got this from
                                    // a NBP call
        (*myABRecord)->atpAddress.aSocket = 15; // socket to send request to
        (*myABRecord)->atpReqCount = 1;  // Size of request data field (disk
                                    //   block requested)
        (*myABRecord)->atpDataPtr = myBuffer;  // Ptr to request to be sent
        (*myABRecord)->atpRspBDSPtr = myBDSPtr;
        (*myABRecord)->atpUserData = 0;  // for your use
        (*myABRecord)->atpXO = FALSE;    // at-least-once service
        (*myABRecord)->atpTimeOut = 5;   // 5-second timeout
        (*myABRecord)->atpRetries = 3;   // 3 retries; request will be sent 4
                                    //   times max
        (*myABRecord)->atpNumBufs = 1;  // We're only expecting 1 block to
                                    //be returned

        async = FALSE;     // Send the request and wait for a response
        myErr = ATPSndRequest (myABRecord, async);
        if (myErr)
            DoError(myErr);
        else {

            // The disk block requested is now in myBuffer.  We can verify
            // that atpNumRsp contains 1, meaning one response received.

        }
    }
}
```