Calling the AppleTalk Manager Using communications features

You can execute many **AppleTalk Manager** routines either synchronously (meaning that the application cannot continue until the routine is completed) or asynchronously (meaning that the application is free to perform other tasks while the routine is being executed).

When an application calls an **AppleTalk Manager** routine asynchronously, an I/O request is placed in the appropriate driver's I/O queue, and control returns to the calling program--possibly even before the actual I/O is completed. Requests are taken from the queue one at a time and processed; meanwhile, the calling program is free to work on other things.

The routines that can be executed asynchronously contain a Boolean parameter called async. If async is TRUE, the call is executed asynchronously; otherwise the call is executed synchronously. Every time an asynchronous routine call is completed, the **AppleTalk Manager** posts a network event. The message field of the event record will contain a handle to the parameter block that was used to make that call.

Note: The reception of network events cannot be relied upon under Multifinder or System 7.

Most **AppleTalk Manager** routines return an integer result code of type OSErr. Each routine description lists all of the applicable result codes generated by the **AppleTalk Manager**, along with a short description of what the result code means. Result codes from other parts of the Operating System may also be returned.

Many high-level language calls to the **AppleTalk Manager** require information passed in a parameter block of the **ATLAPRec**, **ATDDPRec**, **ATNBPRec**, or **ATATPRec**. These structures were formerly unions of a now unused record called the **ABusRecord**. That record no longer exists.

```
// Sending an ALAP packet synchronously and waiting asynchronously for a
// response.
// This example uses the ALTERNATE interface
// NOTE: This code relies on a network event being delivered in order to
// terminate.  The receipt of network events cannot be relied upon under
// Multifinder or System 7
// Assuming inclusion of <MacHeaders>

#include <AppleTalk.h>
#include <string.h>

void myfunction (void);
void DoError (OSErr myErr);
void CheckForMyEvent (void);
main ()
{
    ATLAPRecHandle  myABRecord;
    char            myBuffer[600];  // buffer for both send and
                                    // receive
    Byte            myLAPType;
    short           myErr,
```

```
                    index,
                    dataLen;
Str255              someText;
Boolean             async;

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

else {

    // Call Memory Manager to allocate ATLAPRec

    myABRecord = (ATLAPRecHandle) NewHandle(lapSize);
    myLAPType = 73;

    // Enter myLAPType into protocol handler table and install default
    // handler to service frames of that ALAP type.  No packets of
    // that ALAP type will be received until we call LAPRead.

    myErr = LAPOpenProtocol(myLAPType, nil);
    if (myErr)
        DoError (myErr);
    // Have we opened too many protocol types? Remember that DDP
    // uses two of them.

    else {

        // Prepare data to be sent

        strcpy ((char *) someText,
                    "This data will be in the ALAP data area");
        CtoPstr (someText);

        // The .MPP implementation requires that the first two bytes
        // of the ALAP data field contain the length of the data,
        // including the length bytes themselves.

        dataLen = strlen ((char *) someText) + 2;
        myBuffer[0] = dataLen / 256;    // high byte of data length
        myBuffer[1] = dataLen % 256;   // low byte of data length
        for (index = 1; index <= dataLen-2; index++)
            myBuffer[index+1] = someText[index];
        async = FALSE;

        // fill parameters in the ATLAPRec

        (*myABRecord)->lapAddress.dstNodeID = 4;
        (*myABRecord)->lapReqCount = dataLen;
        (*myABRecord)->lapDataPtr = myBuffer;

        // Send the frame

        myErr = LAPWrite (myABRecord, async);
```

```
            // In the case of a sync call, errCode and the abResult field of
            // the myABRecord will contain the same result code.  We can also
            // reuse myABRecord, since we know whether the call has completed.

            if (myErr)
                DoError(myErr);
                // Maybe the receiving node wasn't on-line

            else {

                // We have sent out the packet and are now waiting for a
                // response.  We issue an async LAPRead call so that we don't
                // "hang" waiting for a response that may not come.

                async = TRUE;
                (*myABRecord)->lapAddress.lapProtType = myLAPType;
                // ALAP type we want to receive

                (*myABRecord)->lapReqCount = 600;
                // our buffer is maximum size
                (*myABRecord)->lapDataPtr  = myBuffer;

                myErr = LAPRead (myABRecord, async);     // wait for a packet

                if (myErr)
                    DoError(myErr);
                    // Was the protocol handler installed correctly?

                else {

                        // We can either sit here in a loop and poll the abResult
                        // field or just exit our code and use the event
                        // mechanism to flag us when the packet arrives.

                        CheckForMyEvent(); // your procedure for checking for a
                                           // network event

                        myErr = LAPCloseProtocol(myLAPType);

                        if (myErr)
                            DoError(myErr);
                }
            }
        }
    }
}
```