Making a Connection Listener Establishing a listener

A connection listener is an **AppleTalk DSP (ADSP)** connection end that cannot receive or transmit data streams or attention messages. The sole function of a connection listener is to wait passively to receive an open-connection request and to inform its client, the connection server, when it receives one. The connection server can then accept or deny the open-connection request. If it accepts the request, the connection server selects a socket to use as a connection end, establishes a connection end on that socket, and sends an acknowledgment and connection request back to the requesting connection end. The connection server can use the same socket as it used for the connection listener or can select a different socket as the connection end.

Use the following procedure to establish a connection listener and to use that connection listener to open a connection with a remote connection end:.

1. Use the **MPPOpen** function to open **The .MPP Driver** and then use the **OpenDriver** function to open **The .DSP Driver**. The **OpenDriver** function will always return the reference number for **The .DSP Driver**. You must supply this reference number each time you call **The .DSP Driver**.

2. Allocate nonrelocatable memory for a connection control block (**CCB**). (The **CCB** is described in **Connection Control Block**.

   Connection Control Block. A connection listener does not need send and receive queues or an attention-message buffer. The memory that you allocate becomes the property of **ADSP** when you call the dspCLInit routine to establish a connection listener. You cannot write any data to this memory except by calling **ADSP**, and you must ensure that the memory remains locked until you call the dspRemove routine to eliminate the connection end. The **CCB** is 242 bytes.

3. Call the dspCLInit routine to establish a connection listener. You must provide a pointer to the **CCB**.

   If there is a specific socket that you want to use for the connection listener, you can specify the socket number in the localSocket parameter. If you want **ADSP** to assign the socket for you, specify 0 for the localSocket parameter. **ADSP** returns the socket number when the dspCLInit routine completes execution.

4. If you wish, you can use the **NBPRegister** function to add the name and address of your connection listener to the node's names table.

5. Use the dspCLListen routine to cause the connection listener to wait for an open-connection request. Because the dspCLListen routine does not complete execution until it receives a connection request, you should call this routine asynchronously. You can specify a value for the filterAddress parameter to restrict the network number, node ID, or socket number from which you will accept an open-connection request.

   When the dspCLListen routine receives an open-connection request that meets the restrictions of the filterAddress parameter, it returns a noErr result code (if you executed the routine asynchronously, it places a noErr result code in the ioResult parameter) and places values

in the parameter block for the *remoteCID*, *remoteAddress*, *sendSeq*, *sendWindow*, and *attnSendSeq* parameters.

6. If you want to open the connection, call the dspInit routine to establish a connection end. You can use any available socket on the node for the connection end, including the socket that you used for the connection listener. Because a single socket can have more than one **CCB** connected with it, the socket can function simultaneously as a connection end and a connection listener.

   You can check the address of the remote socket to determine if it meets your criteria for a connection end. Although the *filterAddress* parameter to the dspCLListen routine provides some screening of socket addresses, it cannot check for network number ranges, for example, or for a specific set of socket numbers. If for some reason you want to deny the connection request, call the dspDeny routine, specifying the **CCB** of the connection listener in the *ccbRefNum* parameter. Because the dspCLListen routine completes execution when it receives an open-connection request, you must return to step 5 to wait for another connection request.

7. Call the dspOpen routine to open the connection. Specify the value ocAccept for the *ocMode* parameter and specify in the *ccbRefNum* parameter the reference number of the **CCB** for the connection end that you want to use. When you call the dspOpen routine, you must provide the values returned by the dspCLListen routine for the remoteCID, remoteAddress, sendSeq, sendWindow, and attnSendSeq parameters.

   You can poll the state field in the **CCB** to determine when the connection is open. Alternatively, you can check the result code for the dspOpen routine when the routine completes execution. If the routine returns the noErr result code, then the connection is open.

8. You can now send and receive data and attention messages over the connection, as described in the preceding section entitled, **Opening and Maintaining an ADSP Connection**. When you are ready to close the connection, you can use the dspClose or dspRemove routines, which are also described in the preceding section.

9. When you are finished using the connection listener, you can use the dspCLRemove routine to eliminate it. Once you have called the dspCLRemove routine, you can release the memory you allocated for the connection listener's **CCB**.

The code example below illustrates the use of **ADSP** to establish and use a connection listener. It opens the .MPP and .DSP drivers and allocates memory for the **CCB**. Then it uses the dspCLInit routine to establish a connection listener, uses the Name-Binding Protocol (NBP) to register the name of the connection end on the internet, and uses the dspCLListen routine to wait for a connection request. When the routine receives a connection request, it calls the dspOpen routine to complete the connection.

```
// Using ADSP to establish and use a connection listener
// Assuming inclusion of <MacHeaders>
```

```
#include <AppleTalk.h>
#include <ADSP.h>

void DoError (OSErr myErr);
void GoDoSomething (void);

TPCCB dspCCBPtr;
DSPPBPtr myDSPPBPtr;
MPPPBPtr     myMPPPBPtr;
NamesTableEntry  myNTEName;
short   drvrRefNum;
short   theMppRefNum;
short   connRefNum;
OSErr  myErr;

myErr = OpenDriver("\p.MPP", &theMppRefNum);  // open .MPP driver
if (myErr)     // check and handle error
    DoError(myErr);
myErr = OpenDriver("\p.DSP", &drvrRefNum); // open .DSP driver
if (myErr) // check and handle error
    DoError(myErr);                  // check and handle error

// allocate memory for data buffers
dspCCBPtr = (TPCCB) NewPtr(sizeof(TRCCB));
myDSPPBPtr = (DSPPBPtr) NewPtr(sizeof(DSPParamBlock));
myMPPPBPtr = (MPPPBPtr) NewPtr(sizeof(MPPParamBlock));

// set up dspCLInit parameters

myDSPPBPtr->ioCRefNum = drvrRefNum;  // ADSP driver ref num
myDSPPBPtr->csCode = dspCLInit;
myDSPPBPtr->u.initParams.ccbPtr = dspCCBPtr;  // pointer to CCB
myDSPPBPtr->u.initParams.localSocket = 0;    // local socket number

myErr = PBControl((ParmBlkPtr) myDSPPBPtr, FALSE);
        // establish a connection listener
if (myErr)     // check and handle error
    DoError(myErr);
connRefNum = myDSPPBPtr->ccbRefNum;  // save CCB ref num for later
NBPSetNTE((Ptr) &myNTEName, (Ptr) "\pThe Object", (Ptr) "\pThe Type",
        (Ptr) "\p*", myDSPPBPtr->u.initParams.localSocket);
        // set up NBP names table entry

// set up PRegisterName parameters

myMPPPBPtr->NBP.interval = 7;    // retransmit every 7*8=56 ticks
myMPPPBPtr->NBP.count = 3;  // and retry 3 times
myMPPPBPtr->NBP.NBPPtrs.entityPtr = (Ptr) &myNTEName;
                        // name to register

myMPPPBPtr->NBP.parm.verifyFlag = 0;  // don't verify this name

myErr = PRegisterName(myMPPPBPtr, FALSE);
                            // register this name
if (myErr)                      // check and handle error
```

```
    DoError(myErr);

// set up dspCLListen parameters

myDSPPBPtr->ioCRefNum = drvrRefNum;  // ADSP driver ref num
myDSPPBPtr->csCode = dspCLListen;
myDSPPBPtr->ccbRefNum = connRefNum;  // connection ref num

myDSPPBPtr->u.openParams.filterAddress.aNet = 0;
myDSPPBPtr->u.openParams.filterAddress.aNode = 0;
myDSPPBPtr->u.openParams.filterAddress.aSocket = 0;
// connect with anybody

myErr = PBControl((ParmBlkPtr) myDSPPBPtr, TRUE);

// listen for connection requests

while ( myDSPPBPtr->ioResult == 1 ) {

    // return control to user while waiting for
    // a connection request

    GoDoSomething();
}

if (myErr)                  // check and handle error
    DoError(myErr);

// You received a connection request; now open a connection.
// The dspCLListen call has returned values into the
//  remoteCID, remoteAddress, sendSeq, sendWindow,
// and attnSendSeq fields of the parameter block.

// set up dspOpen parameters

myDSPPBPtr->ioCRefNum = drvrRefNum;  // ADSP driver ref num
myDSPPBPtr->csCode = dspOpen;
myDSPPBPtr->ccbRefNum = connRefNum;  // connection ref num
myDSPPBPtr->u.openParams.ocMode = ocAccept;
                            // open connection mode

myDSPPBPtr->u.openParams.ocInterval = 0;
                            // use default retry interval

myDSPPBPtr->u.openParams.ocMaximum = 0; // use default retry maximum
myErr = PBControl((ParmBlkPtr) myDSPPBPtr, FALSE);
                            // open a connection
if (myErr)
    DoError(myErr);             // check and handle error
```

The Listing in the section
Opening and Maintaining an ADSP Connection shows how to use ADSP to maintain a connection.