Using NBP Information about the Name Binding Protocol

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

On a Macintosh 128K, the **AppleTalk Manager**'s NBP code is read into the application heap when anyone of the NBP (high-level language) routines is called; you can call the **NBPLoad** function yourself if you want to load the NBP code explicitly. when you are finished with the NBP code and want to reclaim the space it occupies, call **NBPUnload**. On a Macintosh 512K or later, the NBP code is read in when the .MPP driver is loaded.

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

When an entity wants to communicate via an AppleTalk network, it should call **NBPRegister** to place its name and internet address in the names table. When an entity no longer wants to communicate on the network, or is being shut down, it should can **NBPRemove** to remove its entry from the names table.

To determine the address of an entity you know only by name, call **NBPLookup**, which returns a list of all entities with the name you specify. Call **NBPExtract** to extract entity names from the list. To create a packed EntityName from text strings specifying the object, type and zone, use **NBPSetEntity**.

If you already know the address of an entity, and want only to confirm that it still exists, call **NBPConfirm**. **NBPConfirm** is more efficient than **NBPLookup** in terms of network traffic.

```
// An example that registers a node as print spooler,
// searches for any print spoolers registered on the network,
// and extracts the information for the first one found.
// Assuming inclusion of <MacHeaders>

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

#define mySocket      20
#define objstr        "\pGene Station"   // We are called "Gene Station"
#define typestr       "\pPrintSpooler"  // and are of type "PrintSpooler"
#define zonestr       "\p*"
#define numExpected 10  // The number of matches expected

void DoError (OSErr myErr);

main ()
{
    ATNBPRecHandle  myABRecord;
    EntityName      myEntity;
    EntityName      searchEntity;
    AddrBlock       entityAddr;
    Ptr             nbpNamePtr;
```

```
OSErr       myErr;
Boolean         async;
short       nbpNameBufSize;
Ptr         myBufPtr;

myErr = MPPOpen();

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

else {
    // Call Memory Manager to allocate ATNBPRec

    myABRecord = (ATNBPRecHandle) NewHandle (nbpSize);

    // Set up our entity name to register

    NBPSetEntity ((Ptr) &myEntity, (Ptr) objstr, (Ptr) typestr,
                (Ptr) zonestr);

    // Allocate data space for the entity name (used by NBP)

    nbpNameBufSize = *objstr + *typestr + *zonestr;
    nbpNamePtr = NewPtr (nbpNameBufSize);

    // Set up the ABusRecord for the NBPRegister call

    (*myABRecord)->nbpEntityPtr = &myEntity;
    (*myABRecord)->nbpBufPtr = nbpNamePtr;
    // Buffer used by NBP internally

    // Socket to register us on
    (*myABRecord)->nbpBufSize = nbpNameBufSize;
    (*myABRecord)->nbpAddress.aSocket = mySocket;

    // Retransmit every 64 ticks
    (*myABRecord)->nbpRetransmitInfo.retransInterval = 8;

    // And try 3 times
    (*myABRecord)->nbpRetransmitInfo.retransCount = 3;

    async = FALSE;
    myErr = NBPRegister(myABRecord, async);

    if (myErr)
        DoError(myErr);
        // Maybe the name is already registered somewhere
        // else on the network

    else {
        // Now that we've registered our name, find others of
        // type "PrintSpooler"

        // Any one of type "PrintSpooler" in our zone
        NBPSetEntity ((Ptr) &searchEntity, (Ptr) "\p=",
```

```
                    (Ptr) "\pPrintSpooler", (Ptr) "\p*");

            // Allocate space for return buffer which will contain
            // a tuple for each match found

            myBufPtr = NewPtr (numExpected * (sizeof (EntityName) +
                    sizeof (AddrBlock) + 4));

            // Buffer to place responses in
            (*myABRecord)->nbpEntityPtr = &searchEntity;
            (*myABRecord)->nbpBufPtr = myBufPtr;

            (*myABRecord)->nbpBufSize = sizeof (* myBufPtr);

            // The field nbpDataField, before the NBPLookup call,
            // represents an approximate number of responses.  After the
            // call, nbpDataField contains the actual number of responses
            // received.

            (*myABRecord)->nbpDataField = numExpected;

            myErr = NBPLookup(myABRecord, async);    // Make sync call
            if (myErr)
                DoError(myErr);
                // Did the buffer overflow?

            else {

                // Get the first reply

                myErr = NBPExtract (myBufPtr,
                    (*myABRecord)->nbpDataField,1,
                    &myEntity, &entityAddr);

                // The socket address and name of the entity are returned
                // here. If we want all of them, we'll have to loop for
                // each one in the buffer.

                if (myErr)
                    DoError(myErr);
                    // Maybe the one we wanted wasn't in the buffer
            }
        }
    }
}
```