AUGUST 1991

Detecting the Presence of IPX, the NetWare Shell, and NetBIOS from a DOS Application

Morgan Adair
Technical Consultant
Systems Engineering Division

Matt Hagen
Technical Consultant
Systems Engineering Division

Before an application tries to use network resources, it should check to see if those resources exist. This AppNote shows how a DOS application can verify that IPX, the NetWare shell, and NetBIOS are present.

Disclaimer

Novell, Inc. makes no representations or warranties with respect to the contents or use of these Application Notes (AppNotes) or of any of the third-party products discussed in the AppNotes. Novell reserves the right to revise these AppNotes and to make changes in their content at any time, without obligation to notify any person or entity of such revisions or changes. These AppNotes do not constitute an endorsement of the third-party product or products that were tested. Configuration(s) tested or described may or may not be the only available solution. Any test is not a determination of product quality or correctness, nor does it ensure compliance with any federal, state, or local requirements. Novell does not warranty products except as stated in applicable Novell product warranties or license agreements.

A DOS Application's Operating Environment

Taken together, DOS, the NetWare client software, and the hardware they run on comprise an operating environment that contains a number of variables. Applications frequently need to be aware of features of the underlying hardware: the amount of memory, the display adapter, the number and type of disk drives. The various versions of DOS add another set of variables to the operating environment. And when a network is added to the equation, the possible combinations of hardware, operating system, and network resources a program may have to operate under can become innumerable.

Naturally, programmers want to take advantage of all the facilities of the operating environment that will add functionality to their products, especially when a program is being developed for commercial release. At the same time, programmers want their software to be compatible with as many combinations of hardware, operating system versions, and network operating system versions as possible.

You probably have encountered programs that make assumptions about the operating environment. For example, many programs assume that the computers they run on have one floppy drive (A:) and one hard drive (C:). Other programs assume that if they send ANSI escape sequences to the console device, they will be handled correctly by the console device driver. Before a program attempts to access network resources, it should be sure that those resources exist. A program that simply assumes that IPX and the NetWare shell are present can hang the workstation it is running on.

The program given in the listing at the end of this AppNote (EXIST.C) illustrates how a program can verify the existence of basic network facilities. As always, source code for this program and the text of this AppNote can be downloaded from CompuServe, Novell Forum A, Forum Library 16, Novell Uploads (GO NOVA). EXIST.C was compiled and tested using Borland C++ v2.0 and Watcom C v8.0. Version 1.2 of the NetWare C Interface for DOS runtime library was used.

Detecting IPX/SPX

The NetWare C Interface provides two routines, IPXInitialize and SPXInitialize, that can be used to verify that IPX and SPX are loaded. IPXInitialize just returns an integer indicating whether IPX is present. SPXInitialize takes four parameters that receive information about the version of SPX that is installed—the major and minor version numbers, the maximum number of connections supported, and the number of available SPX connections. EXIST.C passes NULLs in these parameters, because it is only interested in the value returned by SPXInitialize—if it returns zero, SPX is not installed; if it returns 255, SPX is installed.

If your application needs to know which version of IPX and SPX is installed, it can use functions provided by the NetWare C Interface's Diagnostic Services. Have your application use the Diagnostic Services to send a diagnostic packet to the calling workstation that

returns the IPX and SPX version numbers. The algorithm is as follows:

1. Get the internetwork address of the workstation's network interface card by calling IPXGetInternetworkAddress.

2. Pass the internetwork address to BeginDiagnostics to establish a connection. BeginDiagnostics returns a connection number and a list of network components (IPX/SPX, shell driver, nondedicated file server, and so on) on the target workstation, which in this case is also the calling workstation.

3. Pass the component list to FindComponentOffset to get the offset of the IPX/SPX data in the component list.

4. Pass the offset to GetIPXSPXVersion to get the major and minor version numbers for IPX and SPX from the component list.

5. Call EndDiagnostics to close the connection established by BeginDiagnostics.

Detecting the DOS Shell

To detect the presence of the DOS shell, simply call GetDefault-ConnectionID. If the value returned is zero, the shell has not been loaded. If the shell has been loaded, the function will return the connection ID for the default file server, a value that ranges from 1 to 8. More information on file server connection IDs is given in the section "File Server Connections" below.

GetDefaultConnectionID simply calls DOS function F0h, subfunction 2h. Since this function is not used by DOS, it normally returns zero in the AL register. If the NetWare shell has intercepted the DOS interrupt vector, it returns the connection ID for the default server instead.

Detecting NetBIOS

To detect the presence of NetBIOS or Novell's NetBIOS emulator, an application simply needs to check the value of the interrupt service vector used by NetBIOS. DOS function 35h returns a pointer to the interrupt service routine (ISR) for a specified interrupt. To see if NetBIOS is installed, call DOS function 35h by doing the following:

1. Move 35h to register AH to specify which DOS function to call.

2. Move 5Ch to register AL to specify which interrupt vector to get.

3. Generate a DOS interrupt 21h.

4. Check the value returned in ES.

If the segment value returned in ES is zero or F000h, NetBIOS is not loaded. If any other value is returned, it is a pointer to the segment where NetBIOS's ISR is loaded in memory.

File Server Connections

When loaded into memory, the NetWare shell attempts to establish a connection with a file server. If unsuccessful, the shell unloads itself. If successful, the shell sets up two tables for maintaining its file server connections—the *connection ID table* and the *file server name table*. EXIST.C examines each entry (1-8) in the connection ID table to see if it contains an active connection. If so, it displays the file server name from the corresponding entry in the file server name table and the name of the user logged in on the connection.

Connection ID Table

The connection ID table is an array of eight 32-byte entries. For each connection to a file server, the shell puts the internetwork address of the file server and other connection information into one of the slots in the connection ID table. NetWare interface functions can then refer to a file server connection by the index number of the connection ID table entry corresponding to that file server.

File Server Name Table

The file server name table is an array of eight 48-byte strings. For each file server connection, the shell places the name of the connected server into the slot with the same index number as the corresponding entry in the connection ID table.

Special File Server Connections

The shell can have up to three file server connections that it uses to prioritize where network request packets are sent—the preferred file server, the default file server, and the primary file server.

Preferred File Server. An application can specify that a particular server in the connection ID table is the preferred file server. Until an application or utility tells the shell otherwise, accounting, bindery, and other service requests are directed to the preferred file server.

Default File Server. The default file server is the server corresponding to the default disk drive. If no application has set a preferred file server, the shell directs service requests to the default file server. For example, if the default drive is F:, and the F: drive is mapped to SRD/SYS:APPNOTES, SRD is the default file server.

Primary File Server. The primary file server is usually either the file server the shell first attached to when it was loaded, or the file server that executed the login script. We say usually because a program can call SetPrimaryConnectionID to specify a different primary server. When a preferred file server has not been specified, and the default disk drive is a local drive, the shell directs its requests to the primary file server.

Example Program: EXIST.C

```
/***************************************************************************
* File:     EXIST.C
* Authors: Morgan Adair & Matt Hagen, Novell, Inc.
* Date:     91-06-25
***************************************************************************/

#include <stdio.h>
#include <nit.h>
#include <niterror.h>
#include <nxt.h>
#include <diag.h>
#include <dos.h>

/***************************************************************************
* main
***************************************************************************/

void main(void)
{
    BYTE       componentList[54];
    BeginDiagnosticStruct  networkAddress;
    int     connection;
    int     component;
    AllResponseData    response;
    IPXSPXVersion      responseData;
    int     ccode;
    WORD       connectionNum;
    WORD       connectionID;
    union REGS  regs;
    struct SREGS   sregs;
    char       serverName[48];
    BYTE       majorVer,
               minorVer,
               revLevel;
    char       name[48];


/* Communication Protocols */
/* IPX/SPX present */

    if (IPXInitialize() == IPX_NOT_INSTALLED)
        printf("IPX is NOT loaded.\n");
    else {
        printf("IPX IS loaded.\n");

        if (SPXInitialize(NULL, NULL, NULL, NULL) == SPX_NOT_INSTALLED)
            printf("SPX is NOT loaded.\n");
        else
            printf("SPX IS loaded.\n");


/* IPX/SPX Version */

        IPXGetInternetworkAddress((BYTE *)&networkAddress);
        if (BeginDiagnostics(&networkAddress, &connectionID, componentList) != SUCCESSFUL)
            printf("Unable to get IPX/SPX versions\n");
        else {
            component = FindComponentOffset(componentList, IPX_SPX_COMPONENT);
            if (component == -1)
                printf("Unable to get IPX/SPX versions\n");
                if (GetIPXSPXVersion(connectionID, component, &response, &responseData)
                          != SUCCESSFUL)
                    printf("Unable to get IPX/SPX versions\n");
                else {
                    printf("IPX Version: %d.%02d\n", responseData.IPXMajorVersion,
                                                     responseData.IPXMinorVersion);
                    printf("SPX Version: %d.%02d\n", responseData.SPXMajorVersion,
                                                     responseData.SPXMinorVersion);
                }
        }
        EndDiagnostics(connectionID);
    }
```

```
/* NetWare DOS Shell */

    connectionID = GetDefaultConnectionID();

    if (connectionID == NULL) {
        printf("The shell is NOT loaded.\n");
        /* if shell is not loaded,
          there is nothing left to live for */
        return;
    } else {
        printf("The shell IS loaded.\n");
        ccode = GetNetWareShellVersion(&majorVer, &minorVer, &revLevel);
        printf("Shell Version: %d.%d, Rev %c\n", majorVer, minorVer, revLevel+'A');
    }


/* NetBIOS */

    regs.h.ah = 0x35;  /* AH = Interrupt 21h Function 35h--
                              Get Interrupt Vector */
    regs.h.al = 0x5C;  /* AL = Interrupt vector to get */
    intdosx(&regs, &regs, &sregs);
    switch (sregs.es) {
        /* ES returns segment of pointer to ISR */
        case 0x0000  :
        case 0xF000  : printf("NetBIOS is NOT loaded.\n");
                       break;
        default      : printf("NetBIOS IS loaded.\n");
    }


/* Connection Information */

    for (connection=1; connection<=8; connection++) {
        if (IsConnectionIDInUse(connection) == 1) {
            SetPreferredConnectionID(connection);
            connectionNum = GetConnectionNumber();
            ccode = GetConnectionInformation(connectionNum, name, NULL, NULL, NULL);
            if (ccode == 0) {
                GetFileServerName(connection, serverName);
                printf("Connection ID %d is logged in to %s as %s.\n",
                                                connection, serverName, name);
            } else
                printf("Connection ID %d is attached only.\n", connection);
        } else
            printf("Connection ID %d is unused.\n", connection);
    }
}
```